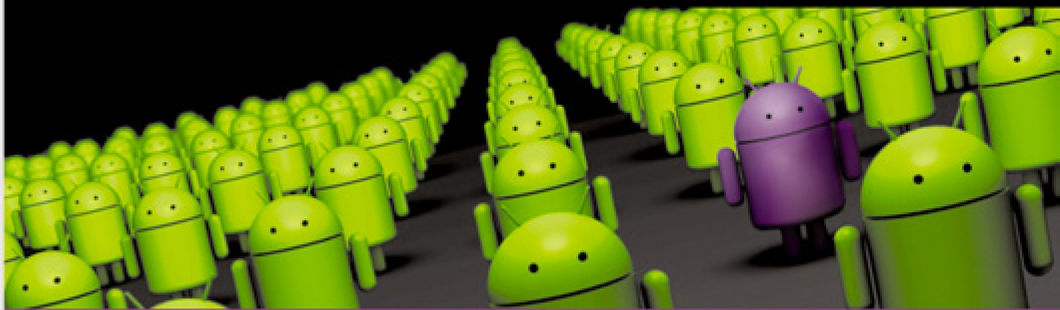
БИБЛИОТЕКА ПРОГРАММИСТА



П. Дейтел Х. Дейтел Э. Дейтел М. Моргано

Android для программистов

Создаём приложения

- Всё, что нужно, для начала разработки приложений для Android
- Лучшие методики разработки Android-приложений
- Взаимодействие с веб-службами Google Maps
- Пошаговое руководство по размещениию приложений на Google Play

PRENTICE HALL

ПИТЕР®

Android™ for Programmers
An App-Driven Approach
Deitel® Developer Series

Paul Deitel
Harvey Deitel
Abbey Deitel
Deitel & Associates, Inc.

Michael Morgano
Imerj

PRENTICE HALL

Upper Saddle River, NJ • Boston• Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Библиотека программиста

П. Дейтел, Х. Дейтел, Э. Дейтел, М. Моргано

Android для программистов

Создаём приложения

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2013

ББК 32.973.2-018.2
УДК 004.451
Д27

П. Дейтел, Х. Дейтел, Э. Дейтел, М. Моргано

Д27 Android для программистов: создаём приложения. — СПб.: Питер, 2013. — 560 с.: ил.

ISBN 978-5-459-01646-8

Приложения Android Market (в настоящее время — Google Play) скачаны уже более миллиарда раз! Эта книга даст вам всё, что нужно, для начала разработки приложений для Android и быстрой публикации их на Android Market. Авторы используют приложение-ориентированный подход, при котором описание каждой технологии рассматривается на примере 16 полностью протестированных приложений для Android. Кроме описания процесса создания приложений, в книге дано пошаговое руководство по размещению ваших приложений на Android Market и примеры успешных публикаций.

Пол Дейтел, Эби Дейтел и Харви Дейтел члены Deitel & Associates Inc. Более миллиона человек во всем мире воспользовались книгами Дейтелов, чтобы освоить Java, C#, C++, C, веб-программирование, JavaScript, XML, Visual Basi c, Visual C++, Perl, Python и др. Майкл Моргано является профессиональным разработчиком Android компании Imerj

ББК 32.973.2-018.2
УДК 004.451

ISBN 978-0132121361 англ.
ISBN 978-5-459-01646-8

Краткое содержание

Содержание

Памяти Даниэля Мак-Кракена (Daniel McCracken). Информационные технологии потеряли в твоем лице одного из выдающихся популяризаторов.

Пол, Харви, Эбби и Майкл

Предисловие

Добро пожаловать в динамичный мир разработки приложений для смартфонов и планшетов Android, в мир Android Software Development Kit (SDK) 2.3.x и 3.x, языка программирования Java™ и интегрированной среды разработки Eclipse™ (IDE).

В этой книге представлены передовые технологии мобильных вычислений, адресованные профессиональным разработчикам ПО. Сердцем книги является *разработка, управляемая приложениями*. Концепции разработки проиллюстрированы на примере *полностью работоспособных приложений Android*, созданных в среде разработки Android. Главы начинаются вводной частью, в которой вкратце описано разрабатываемое приложение. Затем приводятся результаты тестирования приложения и обзор технологий, применяемых в процессе его разработки. После этого выполняется подробный анализ исходного кода приложения. Исходный код всех приложений доступен на сайте www.deitel.com/books/AndroidFP/.

Объемы продаж устройств Android и количество загрузок Android-приложений растут экспоненциально. Мобильные телефоны Android первого поколения появились на рынке в октябре 2008 года. Согласно результатам исследования рынка, проведенного компанией comScore®, в июле 2011 года смартфоны Android занимали 41,8 % рынка смартфонов в США, смартфоны Apple iPhone — 27 %, а BlackBerry — 21,7 %[1]. Количество приложений, загружаемых с Google Play[2], исчисляется миллиардами. Ежедневно активируются более 500 000 устройств Android. Возможности, предоставляемые разработчикам приложений Android, являются поистине безграничными.

По мере того как растет количество пользователей, применяющих для решения повседневных задач смартфоны и планшеты вместо персональных компьютеров, растут и требования к мобильным устройствам. В соответствии с результатами исследований, проводимых компанией comScore, 234 млн американцев использовали мобильные устройства на протяжении трех месяцев, начиная с мая 2011 года; 40,6 % из них загрузили за этот период хотя бы одно приложение[3].

---

[1] www.comscore.com/Press_Events/Press_Releases/2011/8/comScore_Reports_July_2011_U.S._Mobile_Subscriber_Market_Share.

[2] 6 марта 2012 года Android Market был переименован в Google Play. — *Примеч. ред.*

[3] www.comscore.com/Press_Events/Press_Releases/2011/8/comScore_Reports_July_2011_U.S._Mobile_Subscriber_Market_Share.

Ожесточенная конкуренция среди разработчиков популярных мобильных платформ (Android, BlackBerry, iPhone, Palm, Symbian, Windows Phone 7 и других) и мобильных служб приводит к быстрому внедрению инноваций и стремительному обвалу цен. Благодаря соперничеству между десятками производителей устройств Android ускоряется внедрение аппаратных и программных инноваций в сообществе Android. В настоящее время выпускается более 300 различных устройств Android.

Писать эту книгу было очень приятно! Мы узнали и полюбили Android за обилие популярнейших Android-приложений и разнообразие Android-устройств. Нам довелось разработать множество Android-приложений. В процессе разработки рассмотренных в книге приложений вы освоите множество функций и технологий Android, включая аудио, видео, анимацию, телефонию, Bluetooth®, распознавание речи, акселерометр, GPS, компас, виджеты, виджеты приложения, 3D-графику и многое другое. Вы сумеете быстро освоить все, что нужно для начала разработки Android-приложений. А начнем мы с тест-драйва приложения Doodlz, рассматриваемого в главе 1. В процессе чтения главы 3 вы сможете создать свое первое приложение. В главе 2 рассмотрены служба Google Play и основы бизнеса, основанного на разработке и продаже приложений. Вы узнаете, как создаются популярные приложения, научитесь загружать приложения в Google Play и другие онлайновые магазины приложений. Мы расскажем о перспективах бизнеса, связанного с разработкой и продажей приложений, о критериях, в соответствии с которыми приложение относится к категории бесплатных или платных. Здесь же рассматривается маркетинг приложений с помощью Интернет и «сарафанного радио», а также ряд других вопросов.

Авторские права и лицензии на код

Код и Android-приложения, приведенные в книге, являются собственностью компании Deitel & Associates, Inc. Примеры программ, рассмотренные в книге, распространяются на условиях лицензии Creative Commons Attribution 3.0 Unported License (`creativecommons.org/licenses/by/3.0/`). Особо оговорено, что эти примеры не могут использоваться повторно каким-либо образом в учебниках и руководствах, распространяемых в печатном и цифровом формате. Приветствуется использование рассмотренных в книге приложений и их функционала в качестве основы для ваших собственных приложений. Если у вас возникают какие-либо вопросы, обращайтесь по адресу `deitel@deitel.com`.

Целевая аудитория

Предполагается, что читатели этой книги знают язык Java, имеют опыт объектно-ориентированного программирования и знакомы с XML. С другой стороны, в книге используются завершенные рабочие приложения, поэтому, даже не зная Java и XML, но имея опыт объектно-ориентированного программирования на C#/.NET, Objective-C/Cocoa либо C++ (с библиотеками классов), вы сможете быстро освоить излагаемый в книге материал, попутно изучив Java, объектно-ориентированное программирование в стиле Java и XML.

Эта книга не является учебником по Java и XML, но вместе с тем содержит значительный объем материала по этим технологиям в контексте разработки Android-приложений. Если вы хотите глубже освоить Java, обратите внимание на следующие книги:

- *Java for Programmers* (www.deitel.com/books/javafp/).
- *Java Fundamentals*: *Parts I and II LiveLessons videos* (www.deitel.com/books/LiveLessons/).
- *Java How to Program, 9/e* (www.deitel.com/books/jhtp9/).

Особенности книги

Разработка, управляемая приложениями. В каждой из глав, посвященных разработке приложений, представлено одно приложение, рассмотрены функции приложения, приведены снимки экрана выполняющегося приложения, тест-драйв и обзор технологий и архитектуры, используемых при создании приложения. Затем мы строим приложение, представляем его полный исходный код и проводим подробный анализ этого кода; обсуждаем концепции, применяемые в программировании, и демонстрируем функциональные свойства Android API, используемые при создании приложения. В табл. 1 перечислены приложения, представленные в книге, и описаны ключевые технологии, применяемые при их создании.

Таблица 1. Приложения книги и ключевые технологии, используемые для их разработки

Приложение	**Используемые технологии**
Глава 3, приложение Welcome	Dive-Into® Eclipse и ADT
Глава 4, приложение Tip Calculator	Разработка Android-приложений с помощью Java
Глава 5, приложение Favorite Twitter® Searches	Коллекции, виджеты и виды
Глава 6, приложение Flag Quiz	Меню и интенты
Глава 7, приложение Cannon Game	Покадровая анимация и обработка пользовательских событий
Глава 8, приложение Spot-On Game	Анимация с переходами и отслеживание касаний
Глава 9, приложение Doodlz	Графика и акселерометр
Глава 10, приложение Address Book	Адаптеры и AdapterViews
Глава 11, приложение Route Tracker	API приложений Карты и Компас
Глава 12, приложение Slideshow	Доступ к библиотекам Photos и Audio
Глава 13, приложение Enhanced Slideshow	Сериализация объектов и воспроизведение видео
Глава 14, приложение Weather Viewer	Интернет-приложения, веб-службы и виджеты приложений

Android SDK 2.x. В книге рассматривается множество новых функций, включенных в состав набора Android Software Development Kit (SDK) 2.x, в том числе Bluetooth, Google Maps, Camera API, графические API и поддержка различных размеров и разрешений экрана.

Android SDK 3.x для планшетных приложений. Рассматривается новый набор Android SDK 3.x, предназначенный для разработки планшетных приложений. Этот набор поддерживает анимацию, панель действий, фрагменты, уведомления строки состояния и технологию перетаскивания.

Android Maps APIs. Приложение Route Tracker использует Android Maps API, позволяющие использовать в ваших разработках Google™ Maps. Прежде чем приступать к разработке любого приложения, использующего Maps API, следует согласиться с условиями, изложенными в документе Android Maps API Terms of Service (включая условия связанного документа Legal Notices and Privacy Policy), который можно найти на сайте code.google.com/android/maps-api-tos.pdf.

Eclipse. Свободно распространяемая интегрированная среда разработки Eclipse (IDE) вместе со свободно распространяемыми Android SDK и Java Development Kit (JDK) предлагают разработчику все, что нужно для создания и тестирования приложений Android.

Мультимедиа. Приложения используют широкий диапазон мультимедийных возможностей Android, включая графику, изображения, покадровую анимацию, анимацию свойств, аудио, видео, синтез и распознавание речи.

Лучшие методики разработки Android-приложений. В процессе подробного анализа программного кода вашему вниманию будут предложены лучшие методики, применяемые в процессе разработки Android-приложений. Обратите внимание на сайт Android Best Practices Resource Center, находящийся по адресу www.deitel.com/AndroidBestPractices/.

Веб-службы. С помощью веб-служб вы получите возможность воспользоваться богатейшей библиотекой служб, доступных в Интернете, многие из которых являются бесплатными. Приложение Route Tracker, разрабатываемое в главе 11, использует встроенные Android Maps APIs для взаимодействия с веб-службами Google Maps. Разрабатываемое в главе 14 приложение Weather Viewer использует веб-службы WeatherBug's[1].

Соглашения, используемые в книге

Выделение кода. Ключевые сегменты кода в каждой программе выделяются жирным шрифтом.

Использование шрифтов для выделения. Ключевые термины, которые встречаются в тексте книги, выделяются *курсивом*. Элементы интерфейса выделяются шрифтом OfficinaSans (например, меню File), а коды программ на Java и Android — шрифтом Consolas (например, `int x = 5;`).

[1] apireg.weatherbug.com/defaultAPI.aspx.

Использование символа ▸. Символ ▸ применяется для обозначения выбора элемента меню в родительском меню. Например, запись File ▸ New означает, что элемент меню New был выбран в родительском меню File.

Исходный код. Используемый в книге исходный код можно загрузить со следующих сайтов:

```
www.deitel.com/books/AndroidFP/
www.informit.com/title/9780132121361
```

Документация. Документация по разработке на Android и Java, применяемая для создания Android-приложений, доступна на сайте developer.android.com. Документация по интегрированной среде разработки Eclipse находится на сайте www.eclipse.org/documentation.

Темы главы. Каждая глава начинается со списка рассматриваемых в ней тем.

Листинги, таблицы, снимки экрана. В книге приводятся сотни таблиц, листингов исходного кода и снимков экрана Android.

Центры Deitel Online Android Resource Centers

Наши Android Resource Centers включают ссылки на руководства, документацию, загрузки программ, статьи, блоги, подкасты, видео, примеры кода, книги, электронные книги и другие ресурсы. Большая часть перечисленных ресурсов доступна бесплатно. Обратите внимание на следующий постоянно растущий список ресурсов, предназначенных для пользователей Android:

- Android (www.deitel.com/android/);
- Android Best Practices (www.deitel.com/androidbestpractices/);
- Java (www.deitel.com/java/);
- Eclipse (www.deitel.com/Eclipse/);
- SQLite 3 (www.deitel.com/SQLite3/).

Только что появившиеся Resource Center анонсируются в нашем бюллетене новостей Deitel® Buzz Online в Twitter® и Facebook® (см. в следующем разделе).

Бюллетень Deitel & Associates, Inc. Online

Для получения обновлений этой и других книг Deitel, новых и обновленных приложений, сведений о Resource Centers, учебных курсов под руководством инструктора, предложений партнеров и других ресурсов подпишитесь на бесплатно распространяемый бюллетень Deitel® Buzz Online: www.deitel.com/newsletter/subscribe.html.

Следите за нами в Твиттере @deitel и в Фейсбуке www.deitel.com/deitelfan/.

Как связаться с авторами книги

Мы ждем ваши комментарии, критические замечания, исправления и предложения по улучшению книги. Отсылайте корреспонденцию по адресу deitel@deitel.com.

Информацию о замеченных ошибках и уточнения оставляйте по адресу www.deitel.com/books/AndroidFP/, а также в Фейсбуке и Твиттере.

Благодарности

Нам посчастливилось принять участие в работе над этим проектом вместе с командой профессионалов в области издательской деятельности из издательства Prentice Hall/Pearson. Мы высоко ценим экстраординарные усилия и 16-летнее наставничество нашего друга и профессионала Марка Л. Тауба (Mark L. Taub), главного редактора издательской группы Pearson Technology Group. Оливия Бесижио (Olivia Basegio) выполнила огромную работу по привлечению различных участников сообщества пользователей Android и управлению процессом рецензирования. Чати Презертсит (Chuti Prasertsith) разработал обложку книги, проявив чудеса креатива и точности отражения сути книги. Мы объяснили ему концепцию обложки, а он воплотил наш замысел в реальность. Джон Фуллер (John Fuller) выполнил незаурядную работу по менеджированию всех книг из серии Deitel Developer Series books.

Мы хотели бы выразить благодарность нашему другу Ричу Вонгу (Rich Wong) из организации Accel Partners, который снабдил нас ценными контактами участников сообщества разработки мобильных и Android-приложений.

Мы благодарим компанию AWS Convergence Technologies, Inc. — владельца WeatherBug (weather.weatherbug.com/) за то, что она предоставила нам разрешение на доступ к веб-сервисам при создании приложения Weather Viewer, описанного в главе 14.

Выражаем благодарность нашему коллеге Эрику Керну (Eric Kern), который является соавтором нашей другой книги, *iPhone for Programmers*: *An App-Driven Approach*, на материале которой основаны приложения, описанные в настоящей книге.

Рецензенты

Мы хотели бы выразить благодарность нашим рецензентам. В условиях дефицита времени рецензенты внимательно прочли эту книгу и высказали конструктивные предложения, которые повысили точность и целостность изложения материала.

- Пол Бойстерьен (Paul Beusterien), главный специалист компании Mobile Developer Solutions.
- Эрик Дж. Боуден (Eric J. Bowden), главный управляющий компании Safe Driving Systems, LLC.
- Иан Дж. Клифтон (Ian G. Clifton), независимый подрядчик и разработчик приложений Android.
- Даниэль Гэлпин (Daniel Galpin), «Адвокат Андроида» и автор книги *Intro to Android Application Development*.
- Дуглас Джонс (Douglas Jones), старший инженер-программист, компания Fullpower Technologies.
- Себастиан Никопп (Sebastian Nykopp), главный архитектор, компания Reaktor.
- Ронан «Зироу» Шварц (Ronan «Zero» Schwarz), директор по информационным технологиям, компания OpenIntents.

Итак, свершилось! У вас в руках книга «Android для программистов: создаем приложения», благодаря которой вы сможете быстро научиться разрабатывать приложения Android. Мы надеемся, что вы получите удовольствие от чтения книги подобно тому, как мы наслаждались процессом ее написания!

Пол, Харви, Эбби Дейтел и Майкл Моргано, октябрь 2011 г.

Об авторах

Пол Дж. Дейтел (Paul J. Deitel), генеральный директор и главный инженер компании Deitel & Associates, Inc., окончил Массачусетский технологический институт (MIT) по специальности «Информационные технологии» (Information Technology). В Deitel & Associates, Inc. он ведет сотни курсов по Java, C++, C, C#, Visual Basic и программированию в Интернете для корпоративных клиентов, включая Cisco, IBM, Siemens, Sun Microsystems, Dell, Lucent Technologies, Fidelity, NASA (Космический центр имени Кеннеди), National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard Higher Education, Stratus, Cambridge Technology Partners, One Wave, Hyperion Software, Adra Systems, Entergy, CableData Systems, Nortel Networks, Puma, iRobot, Invensys и многих других. Пол и его соавтор, д-р Харви М. Дейтел, являются авторами всемирно известных бестселлеров по языкам программирования, предназначенных для начинающих и для профессионалов.

Харви М. Дейтел (Dr. Harvey M. Deitel), председатель и главный стратег компании Deitel & Associates, Inc., имеет 50-летний опыт работы в области информационных технологий. Он получил степени бакалавра и магистра Массачусетского технологического института и степень доктора философии Бостонского университета. Харви имеет огромный опыт преподавания в колледже и занимал должность председателя отделения информационных технологий Бостонского колледжа. Вместе с сыном — Полом Дж. Дейтелом — он основал компанию Deitel & Associates, Inc. Харви с Полом написали несколько десятков книг и выпустили десятки видеокурсов LiveLessons. Написанные ими книги получили международное призвание и были изданы на японском, немецком, русском, китайском, испанском, корейском, французском, польском, итальянском, португальском, греческом, турецком языках и даже на урду. Дейтел провел сотни семинаров по программированию в крупных корпорациях, академических институтах, правительственных и военных организациях.

Эбби Дейтел (Abbey Deitel), президент компании Deitel & Associates, Inc., закончила школу менеджмента Tepper при университете Карнеги-Мелон и получила степень бакалавра в области промышленного менеджмента. Она курирует коммерческие операции в компании Deitel & Associates, Inc. на протяжении 14 лет. Эбби является автором либо соавтором многочисленных публикаций в Deitel & Associates и вместе с Полом и Харви написала книги *iPhone for Programmers*: *An App-Driven Approach* и *Internet & World Wide Web How to Program, 5/e*.

Майкл Моргано (Michael Morgano), разработчик Android-приложений в компании Imerj™, закончил Северо-Восточный университет, где получил степени бакалавра и магистра в области компьютерных наук. Майкл является соавтором книги *iPhone for Programmers*: *An App-Driven Approach*.

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на веб-сайте издательства: http://www.piter.com.

Подготовительные действия

В этом разделе приведены сведения и инструкции, которые следует внимательно изучить и применить на практике, в результате чего ваш компьютер будет правильно настроен и подготовлен к выполнению упражнений, приведенных в книге. На веб-сайте книги регулярно публикуются дополнения и обновления к этому разделу www.deitel.com/books/AndroidFP/.

Требования к аппаратному и программному обеспечению

Чтобы успешно разрабатывать приложения Android, необходима операционная система Windows®, Linux либо Mac OS X. Для ознакомления с актуальными требованиями к операционной системе посетите сайт developer.android.com/sdk/requirements.html.

Для разработки приложений, описанных в книге, используется следующее программное обеспечение:

- Java SE 6 Software Development Kit;
- Интегрированная среда разработки Eclipse 3.6.2 (Helios) IDE for Java Developers;
- Android SDK, версии 2.2, 2.3.3 и 3.x;
- Плагин для Eclipse ADT (Android Development Tools);

Все эти компоненты подробно рассматриваются в следующих разделах.

Установка Java Development Kit (JDK)

Для разработки Android-приложений требуется набор Java Development Kit (JDK) версии 5 или 6 (JDK 5 или JDK 6). Мы используем JDK 6. Чтобы загрузить JDK для Linux или Windows, перейдите на сайт www.oracle.com/technetwork/java/javase/downloads/index.html.

Для разработки требуется лишь JDK. Инструкции по установке доступны на сайте www.oracle.com/technetwork/java/javase/index-137561.html.

В состав последних версий Mac OS X входит Java SE 6. Проверьте номер версии, выбрав соответствующий параметр в меню Apple.

Установка Eclipse IDE

Для разработки Android-приложений рекомендуется использовать интегрированную среду разработки Eclipse, но вы можете воспользоваться и другими средами разработки, текстовыми редакторами и инструментами командной строки. Для загрузки Eclipse IDE for Java Developers перейдите на веб-страницу www.eclipse.org/downloads/.

С этой веб-страницы можно загрузить последнюю версию Eclipse (3.7.1 на момент написания книги). Чтобы воспользоваться той же версией, которая применялась для создания примеров книги (3.6.2), щелкните на ссылке Older Versions, расположенной над списком загрузок. Выберите версию, подходящую для используемой операционной системы (Windows, Mac или Linux). Чтобы установить Eclipse, необходимо распаковать содержимое архива на жесткий диск. На нашем компьютере с Windows 7 содержимое архива было извлечено в папку C:\Eclipse. За дополнительными сведениями о процессе установки Eclipse обратитесь на сайт bit.ly/InstallingEclipse.

Важно. Чтобы гарантировать корректную компиляцию рассмотренных в книге примеров, сконфигурируйте Eclipse с учетом использования JDK 6, выполнив следующие действия:

1. Найдите папку Eclipse в вашей системе и дважды щелкните на значке Eclipse .
2. После появления окна Workspace Launcher (Запуск рабочей среды) щелкните на кнопке OK.
3. Выполните команду Window ▸ Preferences (Окно ▸ Установки), чтобы отобразить окно Preferences.
4. Раскройте узел Java и выберите узел Compiler (Компилятор). В разделе JDK Compliance (Соответствие JDK) для параметра Compiler compliance level (Уровень соответствия компилятора) выберите значение 1.6.
5. Закройте Eclipse.

Установка Android SDK

Инструментальный набор *Android Software Development Kit (SDK)* включает средства, используемые для разработки, тестирования и отладки приложений Android. Чтобы загрузить Android SDK, обратитесь к сайту developer.android.com/sdk/index.html.

Чтобы загрузить архивный файл SDK, щелкните на ссылке, соответствующей используемой вами платформе: Windows, Mac OS X либо Linux. После завершения загрузки архива распакуйте его содержимое в выбранную на компьютере папку. Набор SDK не включает платформу Android, которая будет загружена отдельно с помощью инструментов Android SDK.

Установка плагина ADT для Eclipse

Плагин Android Development Tools (ADT) для Eclipse обеспечивает возможность использования инструментов Android SDK для разработки приложений Android с помощью интегрированной среды разработки Eclipse. Чтобы установить плагин ADT, обратитесь к веб-сайту developer.android.com/sdk/eclipse-adt.html и *внимательно*

выполните инструкции по загрузке и установке плагина. Если во время установки возникают какие-либо проблемы, воспользуйтесь советами по устранению проблем, которые можно найти на этой же странице.

Установка платформы Android

Теперь установите платформу Android, которая будет использована для разработки приложений. В книге будут использованы версии Android 2.2, 2.3.3 и 3.x. Чтобы установить платформу Android и дополнительные инструменты SDK, выполните следующие действия:

1. Откройте интегрированную среду разработки Eclipse .
2. С помощью появившегося окна Workspace Launcher (Запуск рабочей среды) укажите, где будут храниться приложения, а потом щелкните на кнопке OK.
3. Выполните команды Window ▸ Preferences (Окно ▸ Установки), чтобы отобразить окно Preferences. В этом окне выделите узел Android, а потом в поле SDK Location (Местоположение SDK) укажите путь к папке, в которой будет находиться Android SDK. В нашей системе Windows Android SDK находится в папке c:\android-sdk-windows. Щелкните на кнопке OK.
4. Выполните команды Window ▸ Android SDK Manager (Окно ▸ Диспетчер Android SDK), чтобы отобразить окно Android SDK Manager (рис. 1).

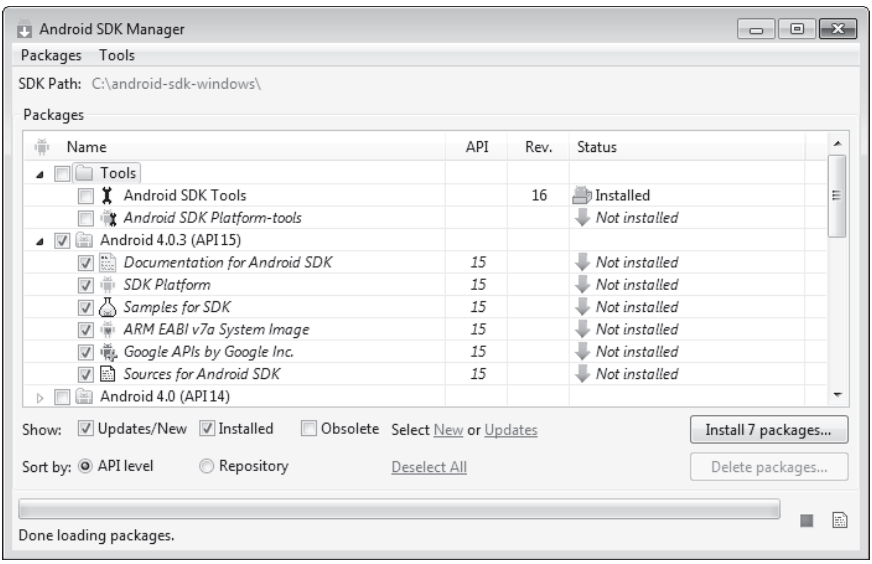


Рис. 1. Окно Android SDK Manager

5. В столбце Name (Имя) отображаются все инструменты, версии платформы Android и расширения, которые можно установить. Чтобы использовать приведенные в этой книге примеры, следует выбрать элементы, показанные на рис. 2.

ПРИМЕЧАНИЕ

Большинство элементов, относящихся к узлу Extras, являются дополнительными. Пакет Google USB Driver package необходим только для тестирования приложений Android на физических Windows-устройствах. Пакет Google Market Licensing package необходим только для разработки приложений, направляющих запросы Google Play, позволяющие определить, имеется ли у пользователя соответствующая лицензия на использование того или иного приложения. Ну и пакет Google Market Billing package обязателен только в тех случаях, если вы намереваетесь продавать цифровой контент с помощью приложения.

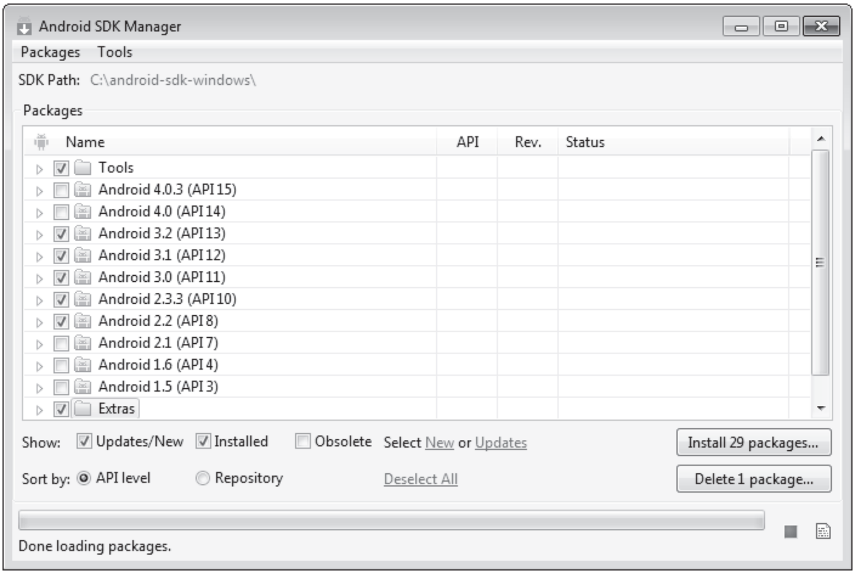


Рис. 2. Выбор устанавливаемых элементов

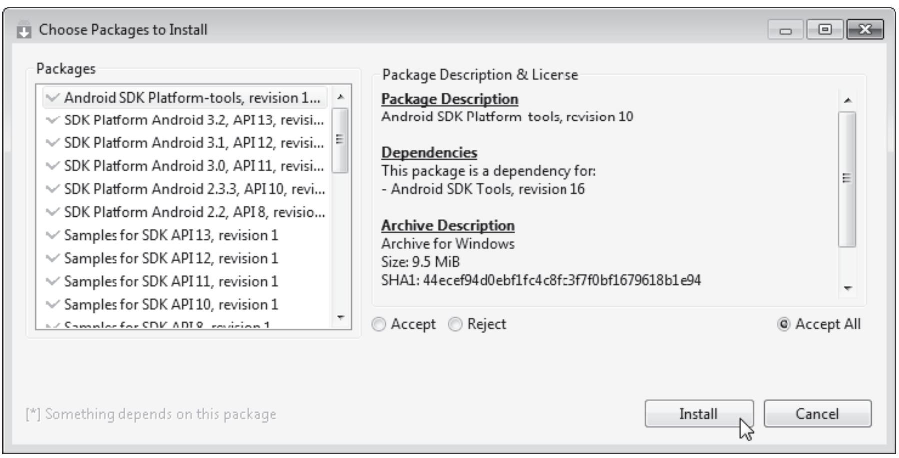


Рис. 3. Окно Choose Packages to Install

6. Щелкните на кнопке Install (Установить) для отображения окна Choose Packages to Install (Выбор пакетов для установки), показанного на рис. 3. В этом окне можно прочесть лицензионное соглашение для каждого элемента. По завершении чтения установите переключатель Accept All (Принять все), потом щелкните на кнопке Install. Текущее состояние процесса установки будет отображаться в окне Android SDK Manager. После завершения установки закройте и повторно откройте Eclipse.

Создание виртуальных устройств Android (AVD) для использования в эмуляторе Android

Эмулятор Android, включенный в состав Android SDK, обеспечивает выполнение приложений Android в эмулированной среде на компьютере, а не на реальном устройстве Android. Прежде чем выполнять приложение с помощью эмулятора, создайте устройство Android Virtual Device (AVD). Это устройство имитирует характеристики реального устройства, для которого разрабатывается приложение. Во время создания этого устройства указывается размер экрана (в пикселях), пиксельная плотность, размер физического экрана, объем карты памяти SD, используемой в качестве хранилища данных, и ряд других параметров. Если нужно протестировать приложения на нескольких устройствах Android, создайте отдельные устройства AVD, эмулирующие уникальные физические устройства. Чтобы создать подобное устройство, выполните следующие действия:

1. Откройте интегрированную среду разработки Eclipse.
2. Выполните команды Window ▸ AVD Manager (Окно ▸ Диспетчер AVD), чтобы отобразить окно Android Virtual Device Manager (Диспетчер виртуальных устройств Android), показанное на рис. 4.
3. Щелкните на кнопке New... (Создать) для отображения окна Create new Android Virtual Device (AVD) (Создание нового виртуального устройства Android), показанного на рис. 5. Затем настройте необходимые параметры и щелкните на кнопке Create AVD (Создать виртуальное устройство Android). Выбранные нами настройки имитируют Android-телефон, используемый нами для тестирования. На момент написания этой книги в нашем распоряжении имелся оригинальный Samsung Nexus S с Android 2.3.3. Для каждого создаваемого устройства AVD настраивается ряд дополнительных параметров, указанных в файле config.ini. Содержимое этого файла пользователь может изменить самостоятельно в соответствии с конфигурацией своего устройства developer.android.com/guide/developing/devices/managing-avds.html.
4. Мы также сконфигурировали устройство AVD, представляющее планшет Motorola Xoom. На этом устройстве установлен Android 3.1, и оно предназначено для тестирования приложений, разработанных для планшетов. Настройки для этого устройства показаны на рис. 6.

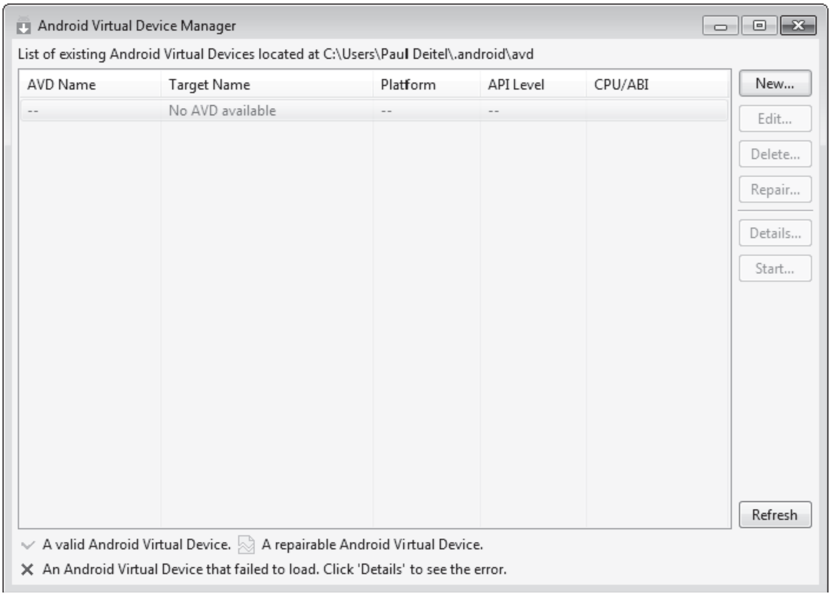


Рис. 4. Окно Android AVD Manager

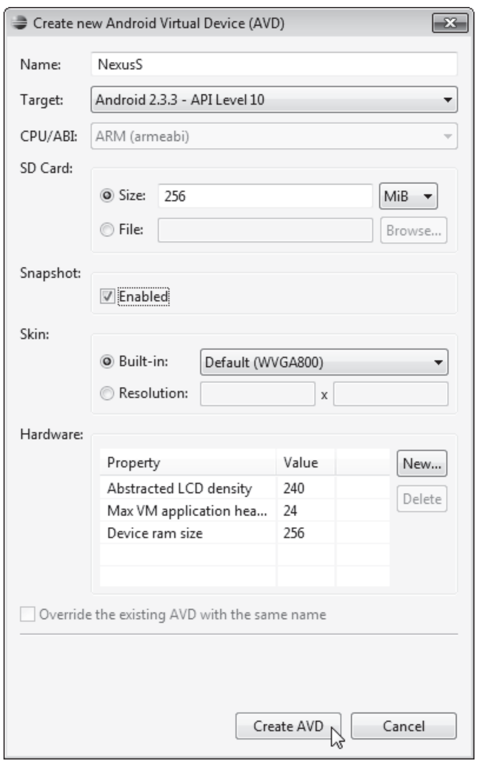


Рис. 5. Окно Create new Android Virtual Device (AVD)

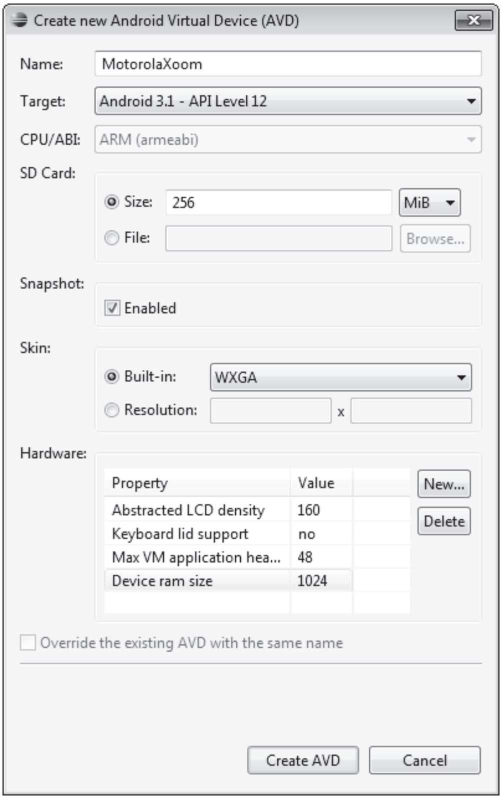


Рис. 6. Окно Create new Android Virtual Device (AVD) на Motorola Xoom

Производительность AVD

На момент написания этой книги производительность устройства AVD была небольшой. Чтобы уменьшить время загрузки AVD, установите флажок Enabled (Включено) в разделе Snapshot (Снимок).

(Дополнительно) Настройка устройства Android для разработки

Как правило, после завершения разработки производится тестирование приложений на физических устройствах Android. Для этого выполните инструкции, приведенные на сайте developer.android.com/guide/developing/device.html.

Если выполняется разработка приложений для платформы Microsoft Windows, понадобится драйвер Windows USB для устройств Android, который (наряду с другими параметрами) выбирается в окне, показанном на рис. 2. В некоторых случаях могут также понадобиться USB-драйверы, специфичные для устройства. Ознакомиться со списком USB-драйверов, предназначенных для различных устройств, можно на сайте developer.android.com/sdk/oem-usb.html.

(Дополнительно) Альтернативные среды разработки приложений Android

Все приложения, приведенные в этой книге, были разработаны с помощью интегрированной среды разработки Eclipse. Несмотря на огромную популярность этой среды для разработки Android-приложений, можно обратиться к другим интегрированным средам и инструментам. Многие опытные разработчики Android-приложений предпочитают использовать инструменты командой строки, а некоторые производители телефонов (например, Motorola) предлагают собственные инструменты разработки Android-приложений. На веб-сайте developer.android.com/guide/developing/projects/projects-cmdline.html приведены сведения, необходимые для разработки Android-приложений с помощью инструментов командной строки. Некоторые из этих инструментов упомянуты в табл. 2.

Таблица 2. Интегрированные среды разработки и инструменты, используемые для разработки Android-приложений

Инструмент	URL-ссылка	Описание
Android	developer.android.com/guide/developing/tools/index.html	Используется для создания, просмотра и удаления устройств AVD; создания и обновления проектов Android; для обновления Android SDK
Android Emulator	developer.android.com/guide/developing/tools/emulator.html	Предназначен для разработки и тестирования приложений Android на компьютере
Android Debug Bridge (adb)	developer.android.com/guide/developing/tools/adb.html	Позволяет управлять состоянием устройства или эмулятора
Apache Ant	ant.apache.org/	Инструмент создания приложений
Keytool and Jarsigner (либо подобный инструмент создания подписи)	developer.android.com/guide/publishing/app-signing.html	Включен в состав JDK. Инструмент Keytool генерирует частный ключ, необходимый для цифровой подписи приложений Android. Инструмент Jarsigner используется для подписи приложений

Где взять примеры кода

Все примеры кода, рассматриваемые в книге, доступны на веб-сайте www.deitel.com/books/androidFP/.

Если вы еще не зарегистрированы на нашем веб-сайте, перейдите на сайт www.deitel.com и щелкните на ссылке Register (Зарегистрироваться), отображенной под логотипом в правом верхнем углу веб-страницы. Введите необходимую информацию. Регистрация абсолютно бесплатна, а введенная вами информация не будет сообщена третьим лицам. Мы рассылаем только сообщения, предназначенные для управления учетной записью,

если вы не подписались дополнительно на бесплатный бюллетень Deitel® Buzz Online на сайте www.deitel.com/newsletter/subscribe.html.

После регистрации на нашем веб-сайте вы получите подтверждающее сообщение электронной почты, содержащее верификационный код, — проверьте корректность адреса электронной почты. Щелкните на верификационной ссылке, находящейся в тексте электронного сообщения, для первичной регистрации на сайте www.deitel.com. Настройте ваш клиент электронной почты на разрешение приема электронных сообщений от deitel.com, чтобы верификационное письмо не попало в папку нежелательной почты.

Затем посетите сайт www.deitel.com и зарегистрируйтесь, щелкнув на ссылке Login (Вход), отображенной под нашим логотипом в левом верхнем углу веб-страницы. Перейдите на сайт www.deitel.com/books/androidFP/. Щелкните на ссылке Examples (Примеры), чтобы загрузить файл Examples.zip на ваш компьютер. Дважды щелкните на файле Examples.zip для распаковки архива.

Теперь вы готовы начать разработку Android-приложений с помощью книги «*Android для программистов*: *создаем приложения*». Наслаждайтесь!

Введение в Android

1

В этой главе...

- История Android и Android SDK.
- Android Market.
- Обзор основных концепций объектной технологии.
- Ключевое ПО, применяемое для разработки Android-приложений, в том числе Android SDK, Java SDK и интегрированная среда разработки Eclipse.
- Основная документация по Android.
- Тест-драйв приложения Android для рисования на экране.
- Центры ресурсов Deitel в Интернете.

1.1. Введение

Добро пожаловать в мир разработки Android-приложений! Мы надеемся, что чтение книги «*Android для программистов*: *создаем приложения*» окажется для вас информативным, интересным, развлекательным и познавательным занятием. Предполагается, что читатели этой книги знают язык Java, имеют опыт объектно-ориентированного программирования и знакомы с XML. С другой стороны, в книге используются завершенные рабочие приложения, поэтому, даже не зная Java и XML, но имея опыт объектно-ориентированного программирования на C#/.NET, Objective-C/Cocoa либо C++ (с библиотеками классов), вы сможете быстро освоить излагаемый в книге материал, попутно изучив Java, объектно-ориентированное программирование в стиле Java и XML, а также овладеть методикой разработки Android-приложений.

Каждая новая технология в книге рассматривается в контексте разработки завершенных рабочих приложений Android, каждое из которых описано в отдельной главе. Мы описываем каждое приложение и тестируем его. Затем вкратце описываются ключевые технологии *Eclipse* (интегрированной среды разработки), Java и *Android SDK (Software Development Kit)*, используемые для создания приложений. Если разрабатываемые приложения используют графический интерфейс, производится его пошаговая

визуальная разработка с помощью Eclipse. Приведены листинги исходного кода с нумерацией строк и выделением ключевых фрагментов кода. Результаты выполнения кода проиллюстрированы одним-двумя экранными снимками. Затем выполняется детальный анализ кода, причем особое внимание уделяется новым концепциям программирования, используемым при разработке приложения. Исходный код всех приложений, рассматриваемых в книге, может быть загружен с веб-сайта www.deitel.com/books/AndroidFP/. В табл. 1.1 перечислена основная документация по Android, которую можно найти в Интернете.

Таблица 1.1. Наиболее важная документация по Android, которую можно найти в Интернете

Название	URL-ссылка
Android Developer Guide	developer.android.com/guide/index.html
Using the Android Emulator	developer.android.com/guide/developing/devices/emulator.htm
Android Package Index	developer.android.com/reference/packages.html
Android Class Index	developer.android.com/reference/classes.html
User Interface Guidelines	developer.android.com/guide/practices/ui_guidelines/index.html
Data Backup	developer.android.com/guide/topics/data/backup.html
Security and Permissions	developer.android.com/guide/topics/security/security.html
Managing Projects from Eclipse with ADT	developer.android.com/guide/developing/projects/projects-eclipse.html
Debugging Tasks	developer.android.com/guide/developing/debug-tasks.html
Tools Overview	developer.android.com/guide/developing/tools/index.html
Publishing Your Apps	developer.android.com/guide/publishing/publishing.html
Android Market Getting Started	market.android.com/support/bin/topic.py?hl=en&topic=15866
Android Market Developer Distribution Agreement	www.android.com/us/developer-distributionagreement.html

Обратитесь к разделу «Подготовительные действия» предыдущей главы, где приведены инструкции по загрузке программ, необходимых для разработки приложений Android. На веб-сайте Android Developer находятся бесплатно загружаемые образцы кода и документация, обучающие видеоролики (табл. 1.21), инструкции по разработке кода и другая полезная информация. Чтобы разместить приложения на «ярмарке

приложений Google» (Google Play), создайте профиль разработчика на веб-сайте play.google.com/apps/publish/signup. Для работы с Google Play вам необходимо уплатить регистрационный взнос $25 через Google Checkout и принять соглашение «Соглашения разработчика о распространении через Google Play». Дополнительные сведения о публикации на Android Market приведены в главе 2.

По мере углубления в разработку приложений Android у вас неизбежно начнут появляться вопросы, касающиеся инструментов, дизайна, безопасности и т. п. В настоящее время существует множество групп новостей и форумов разработчиков Android, где можно поделиться собственными соображениями либо задать вопросы разработчикам (табл. 1.2).

Таблица 1.2. Группы новостей и форумы Android

Название	Подписка	Описание
Android Discuss	Оформите подписку в Google Groups: android- discuss Отправьте письмо по адресу android-discuss subscribe@googlegroups.com	Общая группа дискуссий Android, в которой можно получить ответы на вопросы о разработке приложений
Stack Overflow	stackoverflow.com/questi ons/tagged/android	Этот список рассылки предназначен для начинающих пользователей, здесь они могут задавать вопросы по разработке приложений Android, в том числе вопросы начинающих пользователей по разработке приложений на Java и в среде Eclipse, а также вопросы, касающиеся лучших методик, применяемых в процессе разработки приложений
Android Developers (Разработчики Android-приложений)	Оформите подписку в Google Groups: android- developers Отправьте письмо по адресу android-developerssubscribe@googlegroups.com	Опытные разработчики приложений обмениваются опытом по устранению проблем, возникающих при разработке Android-приложений, проектировании элементов графического интерфейса, оптимизации приложений и прочих проблем, возникающих на этапе разработки и отладки приложений
Android Market Help Forum (Форум поддержки пользователей Android Market)	www.google.com/support/forum/p/Android+market	Коллекция вопросов и ответов по работе с маркетом Android
Android Forums (Форумы Android)	www.androidforums.com/	Коллекция вопросов, ответов и советов по разработке приложений на платформе Android, предназначенных для различных Android-устройств

1.2. Обзор платформы Android

Смартфоны, реализованные на платформе Android первого поколения, появились на рынке в октябре 2008 года. Согласно результатам исследований рынка, выполненных компанией Gartner[1], мировые продажи смартфонов на базе Android выросли в девять раз в первом квартале 2010 года по сравнению с первым кварталом предыдущего года. Результаты исследования рынка, выполненные компанией Nielsen[2], показали, что к марту 2011 года рыночная доля смартфонов Android составила уже 37 % от общего рынка смартфонов, продаваемых в США, тогда как рыночная доля смартфонов Apple iPhone составила 27 %, а рыночная доля смартфонов BlackBerry — 22 %. В августе 2010 года[3] ежедневно выполнялась активация более 200 000 смартфонов Android, тогда как всего лишь двумя месяцами раньше активировалось не более 100 000 устройств Android ежедневно. А уже в июне 2011 года ежедневно активировалось более 500 000 устройств Android. В настоящее время на мировом рынке доступно более 300 различных гаджетов Android.

Операционная система Android была разработана компанией Android, Inc. В июле 2005 года акции этой компании приобрела корпорация Google. В ноябре 2007 был сформирован консорциум Open Handset Alliance™, который изначально объединил 34 компании, а в настоящее время он объединяет уже 81 компанию (www.openhandsetalliance.com/oha_members.html). В задачи этого консорциума входит разработка приложений для платформы Android, внедрение инноваций в технологии, применяемые для разработки мобильных гаджетов, а также повышение удобства работы с устройствами Android при одновременном снижении затрат на производство и снижение цен для потребителей. На основе Android создано множество моделей смартфонов, электронных книг и планшетных компьютеров.

Открытость платформы и открытый исходный код

Одно из главных преимуществ платформы Android — ее открытость. Операционная система Android построена на основе *открытого исходного кода* и распространяется на свободной основе. Это позволяет разработчикам получить доступ к исходному коду Android и понять, каким образом реализованы свойства и функции приложений. Любой пользователь может принять участие в совершенствовании операционной системы Android. Для этого достаточно отправить отчет об обнаруженных ошибках (на веб-странице source.android.com/source/report-bugs.html) либо принять участие в одной из групп дискуссий Open Source Project (source.android.com/community/index.html). В Интернете доступны различные приложения Android с открытым исходным кодом, предлагаемые компанией Google и рядом других производителей (табл. 1.3). В табл. 1.4 показано, где можно получить исходный код Android, каким образом освоить философию, которая лежит в основе операционной системы с открытым кодом, и получить лицензионную информацию.

[1] www.gartner.com/it/page.jsp?id=1372013.

[2] blog.nielsen.com/nielsenwire/online_mobile/u-s-smartphone-market-whos-the-mostwanted/.

[3] www.wired.com/gadgetlab/2010/08/google-200000-android-phones/.

Таблица 1.3. Ресурсы приложений Android с открытым исходным кодом

Описание	**URL-ссылка**
Обширный список приложений с открытым исходным кодом, разбитых по категориям (игры, утилиты и пр.)	En.wikipedia.org/wiki/List_of_open_source_Android_applications
Примеры приложений Google для платформы Android	code.google.com/p/apps-for-android/
Тридцать примеров приложений, демонстрирующих различные свойства платформы Android	developer.android.com/resources/browser.html?tag=sample
Двенадцать приложений Android с открытым исходным кодом	www.techdrivein.com/2010/11/12-open-sourceandroid-applications.html
Ссылки на игры Android с открытым исходным кодом	www.techdrivein.com/2010/12/15-nice-andsimple-open-source-android.html

Таблица 1.4. Ресурсы, включающие исходный код Android и документацию

Описание	**URL-ссылка**
Исходный код Android	source.android.com/source/download.html
Философия и цели разработчиков платформы Android	source.android.com/about/philosophy.html
Лицензии	source.android.com/source/licenses.html
Часто возникающие вопросы и ответы на них (FAQ)	source.android.com/faqs.html#aosp

Java

При разработке приложений Android используется язык программирования Java, являющийся одним из наиболее распространенных. Использование Java стало логичным выбором для платформы Android, потому что это мощный, свободный и открытый язык. На Java разрабатываются полномасштабные корпоративные приложения, расширяется функционал веб-серверов, создаются приложения, предназначенные для пользовательских устройств (мобильных телефонов, пейджеров и персональных цифровых помощников), и это далеко не полный перечень возможных областей его применения.

Java является кроссплатформенным языком. Он позволяет разрабатывать приложения, на зависящие от аппаратных особенностей того или иного устройства. Опытные программисты на языке Java могут быстро освоить создание приложений для платформы Android с помощью Android API и других вспомогательных средств, предлагаемых независимыми производителями.

Открытость платформы способствует быстрому внедрению инноваций. Свыше десяти OEM-производителей из 48 стран, получивших лицензию на производство Android-устройств в 59 странах, выпускают гаджеты на основе платформы Android[1]. Естественно, что все они конкурируют между собой, что идет на пользу конечному потребителю.

Язык программирования Java является объектно-ориентированным и предоставляет разработчикам возможность получить доступ к мощным библиотекам классов, ускоряющих разработку приложений. Программирование графического интерфейса пользователя является управляемым событиями. В частности, в этой книге приведены примеры приложений, которые реагируют на инициируемые пользователями события, такие как касания экрана или нажатие кнопок. Помимо непосредственного написания кода приложений, можно воспользоваться визуальным конструктором среды Eclipse, позволяющим собирать графический интерфейс из готовых объектов, таких как кнопки и текстовые поля, перетаскивая их в определенные места экрана, добавляя подписи и изменяя их размеры. С помощью интегрированной среды разработки Eclipse и подключаемого модуля ADT (Android Development Tools) можно создавать, тестировать и отлаживать приложения Android, а также заниматься проектированием пользовательского интерфейса.

Мультисенсорный экран

Многие современные смартфоны Android сочетают в себе функции мобильных телефонов, интернет-клиентов, MP3-плееров, игровых консолей, цифровых фотоаппаратов и многого другого. Эти устройства упакованы в компактные корпусы и оборудованы полноцветными *мультисенсорными* экранами. Подобный экран позволяет управлять устройством с помощью *касаний* и *жестов*, как показано в табл. 1.5.

Таблица 1.5. Жесты, применяемые на устройствах Android

Название	Физическое действие	Применение
Касание	Коснитесь один раз экрана	Открытие приложения, «нажатие» кнопки или элемента меню
Двойное касание	Дважды коснитесь экрана	Увеличение или уменьшение масштаба просмотра изображений, карт Google Maps и веб-страниц
Длинное нажатие	Нажмите выбранную область экрана и удерживайте палец в этой позиции	Открытие контекстного меню либо перемещение в области экрана пиктограмм приложений или других объектов
Перетаскивание	Нажмите пальцем и перетащите его вдоль экрана	Перемещение объектов или пиктограмм либо точная прокрутка веб-страницы или списка

продолжение ⇨

[1] code.google.com/events/io/2010/.

Таблица 1.5 *(продолжение)*

Название	Физическое действие	Применение
Смахивание	Нажмите и быстро смахните пальцем вдоль экрана в нужном направлении	Прокрутка содержимого объекта ListView (список контактов) либо DatePickerView и TimePicker View (даты и время календаря)
Масштабирование двумя пальцами	Коснитесь экрана двумя пальцами, а потом сведите или разведите их для изменения масштаба	Увеличение или уменьшение масштаба просмотра экрана (увеличение или уменьшение текста и рисунков)

Мультисенсорный экран упрощает переключение между телефоном, приложениями, музыкальной библиотекой, веб-браузером и т. п. На экране может отображаться клавиатура, предназначенная для ввода сообщений электронной почты и данных приложений (некоторые устройства Android также снабжены физическими клавиатурами). С помощью двух пальцев можно увеличивать масштаб просмотра (путем разведения пальцев) или уменьшать масштаб просмотра (с помощью сведения пальцев) фотографий, видеороликов и веб-страниц. Путем смахивания пальцем экрана по вертикали или по горизонтали можно выполнять прокрутку содержимого этого экрана.

Встроенные приложения

В комплект поставки устройств Android входят различные встроенные приложения, набор которых зависит от устройства. Обычно это приложения Телефон (Phone), Контакты (Contacts), Почта (Mail), Браузер (Browser) и ряд других приложений. Многие производители модифицируют стандартные приложения. В дальнейшем будет продемонстрировано взаимодействие пользователя с приложениями независимо от того, каким образом они были изменены производителем.

Соглашения относительно именования версий Android

Каждая новая версия Android получила наименование от определенного десерта (в алфавитном порядке):

- Android 1.6 (Donut);
- Android 2.0–2.1 (Eclair);
- Android 2.2 (Froyo);
- Android 2.3 (Gingerbread);
- Android 3.0 (Honeycomb).

1.3. Android 2.2 (Froyo)

Версия Android 2.2, которая также называется «Froyo» (замороженный йогурт), появилась в мае 2010 года. В состав этой версии были включены целый ряд новых функций и усовершенствований (табл. 1.6). В следующих разделах главы будут рассмотрены более новые версии Android 2.3 (Gingerbread, имбирный пряник) и Android 3.0 (Honeycomb, медовые соты).

Таблица 1.6. Пользовательские функции Android 2.2 (developer.android.com/sdk/android-2.2-highlights.html)

Функция	Описание
Улучшенное управление памятью и увеличенная производительность	Обновления: • Усовершенствование Dalvik Virtual Machine привело к увеличению производительности в 2–5 раз по сравнению с Android 2.1. • Механизм Chrome V8 ускорил загрузку веб-страниц, содержащих сценарии JavaScript. • Управление памятью ядра позволило улучшить производительность устройств
Автообнаружение	Пользователи Exchange могут быстро синхронизировать учетные записи Exchange с устройствами Android путем указания имени пользователя и пароля
Календарь	Пользователи могут синхронизировать календари Exchange Calendar с приложением `Календарь` (Calendar)
Поиск в Global Address Lists (GAL, Глобальные списки адресов)	Доступ к адресам пользователей электронной почты и списков рассылки в пользовательских системах электронной почты Microsoft Exchange, возможность автозавершения имен контактов получателей создаваемых сообщений электронной почты
Пароли	Пользователи могут создавать пароли, состоящие из букв и цифр, с помощью которых блокируются устройства. В результате предотвращается доступ к заблокированным устройствам и существенно повышается степень безопасности
Удаленная очистка	Если вы потеряли устройство Android, воспользуйтесь функцией `Remote Wipe` (Удаленная очистка). Эта функция выполняет сброс устройства к заводским установкам, удаляя при этом все пользовательские данные. Благодаря этому ваши самые личные данные не попадут в руки злоумышленника. При отправке команды удаленной очистки (обычно при помощи специального SMS-сообщения) все ваши данные, которые не были скопированы на резервные носители, будут удалены. Доступность функции Remote Wipe зависит от производителя устройства и диспетчеров политики устройства.
Контакты и учетные записи	Функция `Quick Contact` (Быстрый контакт) для устройств Android обеспечивает упрощенный доступ к контактной информации и режимам обмена данными с контактами (например, по электронной почте, с помощью SMS-сообщений или по телефону). Пользователь может коснуться фотографии контакта (например, в списке контактов, галерее фотографий, в окне клиента электронной почты или в календаре), вызвать виджет `Quick Contact` с различными коммуникационными режимами. Виджет Quick Contact можно включать в состав различных приложений, создаваемых разработчиками

продолжение ☞

Таблица 1.6 *(продолжение)*

Функция	Описание
Камера	Элементы управления камерой в Android 2.2 включают функции контроля вспышки и цифрового масштабирования (зума) камеры. Пользователи могут изменять настройки камеры, учитывая условия внешней среды (ночь, солнечный свет, быстрое перемещение), добавлять различные эффекты (сепия, красный либо синий оттенок) и вносить ряд других изменений. Можно запрограммировать предварительный просмотр, настройки съемки, установки выборки и кодирования видео
Виртуальная клавиатура Android	Значительно улучшена раскладка клавиатуры. В результате существенно облегчен ввод данных с помощью мультисенсорного экрана. Теперь вы не пропустите нужные клавиши, даже если будете работать с клавиатурой двумя пальцами
Улучшенный словарь	Интеллектуальный словарь учитывает статистику использования слов пользователем и в варианты предлагаемых слов включает контакты пользователя
Браузер	Усовершенствованный интерфейс браузера включает новую строку адреса, которая обеспечивает выполнение поиска и навигации после ее касания пальцем. С помощью двойного касания веб-страниц осуществляется изменение их масштаба просмотра. Включена поддержка HTML5, обеспечивающего возможность просмотра видео и перетаскивания. Ранее эти возможности были доступны лишь с помощью подключаемых модулей, предлагаемых независимыми поставщиками, например Adobe Flash. (*Примечание*. Браузер также поддерживает Flash.)
Многоязычная клавиатура	Пользователи могут добавлять раскладки клавиатур для других языков и легко переключаться между ними. Чтобы сменить раскладку, «смахните» клавишу пробела клавиатуры в направлении справа налево. Чтобы добавить клавиатуру (для устройства Android или для эмулятора), выполните команды Settings ▸ Language & keyboard ▸ Android keyboard ▸ Input languages (Настройки ▸ Язык и клавиатура ▸ Клавиатура Android ▸ Языки ввода)
Медиафреймворк	Медиафреймворк Android Stagefright обеспечивает воспроизведение видео и поддерживает прогрессивные HTTP-потоки — отсылка видео браузеру через Интернет с помощью протокола HyperText Transfer Protocol (Протокол передачи гипертекста) и воспроизведение видео даже в процессе его загрузки. Также в Android поддерживается предыдущая версия Медиафреймворк, OpenCORE

Функция	**Описание**
Bluetooth	Пользователи могут подключать Android-устройства к другим устройствам Bluetooth, таким как гарнитуры и автомобильные док-станции (предназначены для подключения телефона к автомобильной системе «свободные руки»), передавать сведения о контактах телефонам с поддержкой Bluetooth и выполнять голосовой набор номеров
WiFi-тетеринг	В Android 2.x встроена поддержка использования мобильного телефона в качестве точки доступа других устройств к интернету. При этом мобильный телефон выступает в качестве модема и маршрутизатора. Подключение устройств к мобильному телефону может осуществляться как по беспроводному, например WiFi, так и по проводному каналу, например через USB-порт. См. www.engadget.com/2010/05/13/android-2–2-froyo-toinclude-usb-tethering-wifi-hotspot-funct/

Новые функции Android 2.2, предназначенные для разработчиков

С помощью службы *C2DM (Android Cloud to Device Messaging, обмен сообщениями с устройствами через облако)* разработчики приложений могут пересылать данные со своих серверов на приложения, установленные на устройствах Android, даже если эти приложения в данный момент не активны. Сервер оповещает приложения о непосредственном подключении к нему для приема обновления или пользовательских данных[1]. Путем регистрации в общедоступной учетной записи Android Market можно получить доступ к *отчетам об ошибках приложений Android (Android Application Error Reports)*, в которых находятся отчеты пользователей, создаваемые после аварийного завершения или «зависания» приложений.

В Android 2.2 также появились несколько новых библиотек API, упрощающих добавление различных функциональных свойств в приложения (табл. 1.7). Некоторые из этих новых фреймворков будут использованы в книге. Также используются веб-службы, с помощью которых создаются *мэшапы*. В этом случае обеспечивается возможность быстрой разработки приложений путем комбинирования веб-служб, используемых в различных организациях, с различными типами вводимой информации (например, RSS, Atom, XML, JSON и др.) (табл. 1.8). Например, на сайте www.housingmaps.com веб-службы применяются для комбинирования списков объектов недвижимости Craigslist (www.craigslist.org) с картами Google Maps (библиотеки API, которые чаще всего применяются при создании мэшапов). В результате пользователь может просмотреть на карте местоположение выбранного дома или квартиры. Веб-службы WeatherBug будут рассматриваться в главе 14.

[1] code.google.com/android/c2dm/.

Таблица 1.7. Библиотеки API, доступные в Android 2.2 (developer.android.com/sdk/android-2.2-highlights.html)

Категория API	Описание
Размещение приложений во внешней памяти (Apps on external storage)	Приложения, которые могут храниться во внешней памяти устройства Android, а не во внутренней памяти (по умолчанию)
Камера и камкордер (Camera and camcorder)	Новые функции и возможности, в том числе у API Camera Preview: удвоенная частота смены кадров (20 кадров в секунду), наличие портретной ориентации, элементы управления масштабированием, наличие данных экспозиции и утилиты миниатюр. Новые классы CamcorderProfile применяются в приложениях для определения наличия в пользовательском устройстве реализованного на аппаратном уровне камкордера
Резервирование данных (Data backup)	Резервирование данных в «облачном» хранилище с последующим восстановлением после аппаратного сброса устройства к исходным заводским настройкам либо при аварийном восстановлении данных
Политика управления устройствами (Device policy management)	Создание приложений администратора, предназначенных для управления уровнем безопасности устройства (например, длиной пароля)
Графика (Graphics)	Набор графических API, обеспечивающих доступ к OpenGL ES 2.0. Ранее эти API были доступны только с помощью Android NDK — набора инструментов, обеспечивающего использование собственного кода для компонентов приложений, которые являются критическими к производительности (developer.android.com/sdk/ndk/overview.html)
Медиафреймворк (Media framework)	Набор API, обеспечивающих выбор аудиопотока, автоматическое сканирование файлов в базе данных медиаресурсов (аудио- и видеофайлы), обнаружение завершения загрузки звукового файла, автоматическая пауза и продолжение воспроизведения звука и ряд других возможностей
Фреймворк интерфейса пользователя (UI framework)	Элементы управления режимами (автомобильным, ночным и режимом настольного компьютера) UiModeManager позволяют настраивать пользовательский интерфейс приложения. Библиотека API детектора жестов масштабирования обеспечивает улучшенную обработку событий, а расположенная в нижней части экрана лента TabWidget теперь является настраиваемой

Таблица 1.8. Некоторые популярные веб-службы (www.programmableweb.com/apis/directory/1?sort=mashups)

Веб-службы	Применение
Google Maps	Картографические службы
Facebook	Социальные сети
Foursquare	Мобильный контроль
LinkedIn	Социальные бизнес-сети
YouTube	Поиск видео
Twitter	Микроблоги
Groupon	Социальная коммерция
Netflix	Аренда фильмов
eBay	Интернет-аукционы
Wikipedia	Коллективная энциклопедия
PayPal	Платежи
Last.fm	Интернет-радио
Amazon	Электронная коммерция: продажа книг, устройств чтения электронных книг и т. д.
Salesforce.com	Управление отношениями с клиентами и партнерами
Skype	Интернет-телефония
Microsoft	Поисковая машина Bing
Flickr	Публикация фотографий в Интернете
Zillow	Прайс-листы на недвижимость
Yahoo!	Поисковая машина Yahoo!
WeatherBug	Прогноз погоды

В табл. 1.9 перечислены каталоги с информацией о большинстве популярных веб-служб.

Таблица 1.9. Каталоги веб-служб

Каталог	URL-ссылка
ProgrammableWeb	www.programmableweb.com
Webmashup.com	www.webmashup.com/
Webapi.org	www.webapi.org/webapi-directory/
Google Code API Directory	code.google.com/apis/gdata/docs/directory.html
APIfinder	www.apifinder.com/

1.4. Android 2.3 (Gingerbread)

Версия Android 2.3 (Gingerbread — «имбирный пряник»), появившаяся в декабре 2010 года (субверсия Android 2.3.3 с незначительными обновлениями, появилась в феврале 2011), предлагает усовершенствованные пользовательские функции, например более удобную клавиатуру, улучшенные навигационные функции, более эффективное использование аккумуляторной батареи и ряд других возможностей. Некоторые новые и наиболее важные пользовательские функции и обновления приведены в табл. 1.10.

Таблица 1.10. Пользовательские функции Android 2.3 (developer.android.com/sdk/android-2.3-highlights.html)

Функция	Описание
Управление электропитанием	Приложения, которые используют процессорные ресурсы при работе в фоновом режиме, или находятся в неактивном состоянии дольше, чем обычно, могут быть при необходимости принудительно закрыты Android для экономии энергии аккумулятора и повышения производительности системы. Пользователи также могут просматривать список приложений и компонентов системы, потребляющих энергию аккумулятора
Ярлык Manage Applications	С помощью ярлыка Manage Applications (Управление приложениями), доступного в меню Options (Параметры) на экране Home (Главный экран), пользователи могут просматривать все выполняемые приложения. Для каждого приложения можно узнать объем используемого хранилища и памяти, предоставленные приложению разрешения (определяют возможность чтения контактных данных пользователя, создания подключений Bluetooth и других задач), а также выполнить ряд других задач. Пользователи также могут «принудительно завершить выполнение» приложения
Near-field communications (Коммуникации ближнего поля)	Near Field Communication, NFC (коммуникация ближнего поля) — технология беспроводной высокочастотной связи малого радиуса действия, которая дает возможность обмена данными между устройствами, находящимися на расстоянии около 10 сантиметров. Эта технология — простое расширение стандарта бесконтактных карт, которая объединяет интерфейс смарткарты и считывателя в единое устройство. Устройство NFC может поддерживать связь и с существующими смарткартами и считывателями, и с другими устройствами NFC и таким образом совместимо с существующей инфраструктурой бесконтактных карт, уже использующейся в общественном транспорте и платежных системах. NFC нацелена, прежде всего, на использование в мобильных телефонах[1]
Улучшенные функции копирования и вставки	Вы можете коснуться слова, чтобы выделить его, перетащить маркеры для настройки выделенной области, затем скопировать текст, нажав на выделенную область, после чего вставить текст. Можно также перемещать курсор путем перетаскивания стрелки курсора

Функция	Описание
Камера	Приложения могут получить доступ как к фронтальной, так и к тыловой камере либо сразу же к обеим камерам
Голосовые вызовы через Интернет	В Android поддерживается протокол Session Initiation Protocol (SIP, Протокол инициализации сеанса) и протокол стандарта Internet Engineering Task Force (IETF). С помощью этих протоколов обеспечивается инициализация и завершение голосовых вызовов, осуществляемых через Интернет. Пользователи, обладающие учетными записями SIP (созданными с помощью инструментов от независимых поставщиков), могут совершать голосовые вызовы других пользователей, имеющих учетные записи SIP, через Интернет. Обратите внимание, что поддержка SIP и вызовов через Интернет присуща далеко не всем устройствам Android. Чтобы получить список SIP-провайдеров, обратитесь к веб-сайту www.cs.columbia.edu/sip/service-providers.html
Приложение Downloads	Пользователи могут получать доступ к файлам, загруженным с электронной почты, браузера или других мест, с помощью приложения Downloads (Загрузки)

В состав платформы также включены многочисленные новые функции для разработчиков, предназначенные для расширенных коммуникаций, разработки игр и мультимедийных приложений (табл. 1.11). Дополнительные сведения о каждой из этих функций можно найти на веб-сайте developer.android.com/sdk/android-2.3-highlights.html.

Таблица 1.11. Функции Android 2.3, предназначенные для разработчиков (developer.android.com/sdk/android-2.3-highlights.html)

Функция	Описание
Интернет-телефония	Благодаря появившейся поддержке SIP можно встраивать в приложения функции интернет-телефонии (возможность создания и приема звонков)
Библиотека API Near-field communications	Встроены приложения, применяемые для считывания данных из тегов либо устройств NFC и генерирования ответных данных. С помощью приложений Android 2.3.3 можно также записывать теги и работать в одноранговом режиме с другими устройствами. Обратите внимание, что степень поддержки NFC будет различной для разных устройств
Библиотека API Audio effects	Добавление эффектов эквалайзера (настройка усиления в области низких или высоких частот), усиления басов (увеличение громкости звучания басового звука), виртуализации наушников (имитируется пространственное звучание) и реверберации (эхо-эффектов) в одну звуковую дорожку или несколько дорожек

продолжение ☞

Таблица 1.11 *(продолжение)*

Функция	Описание
Новые аудиоформаты	Встроена поддержка кодеков Advanced Audio Coding (AAC, Расширенный звуковой кодек — «наследник» MP3) и Adaptive Multi-Rate Wideband (AMRWB, Адаптивный широкополосный мультичастотный кодек), обеспечивающих получение звука высокого качества
Новые видеоформаты	В открытый контейнерный формат WebM встроена поддержка открытого алгоритма видеосжатия VP8
Библиотека API Camera	С помощью библиотеки API Camera, обладающей расширенным набором функций, можно получить доступ к фронтальной и тыловой камере устройства, определить используемые функции и активизировать соответствующую камеру

1.5. Android 3.0 (Honeycomb)

Согласно некоторым оценкам, к 2015 году доля продаваемых планшетов составит более 20 % от всего рынка продаваемых персональных вычислительных устройств[1]. Среди существующих планшетов быстрее всего растет доля продаваемых планшетов Android. Согласно сведениям, опубликованным компанией Consumer Electronic Show, в 2011 году было анонсировано 85 новых планшетов Android[2]. В версии Android 3.0 (Honeycomb, «пчелиные соты») появились усовершенствования пользовательского интерфейса, предназначенные для устройств с большим экраном (то есть планшетов). Среди этих улучшений — усовершенствованная клавиатура, обеспечивающая более комфортный ввод данных, 3D-имитация пользовательского интерфейса, панели System (Система) и Action (Действие), облегчающие навигацию, и ряд других улучшений (табл. 1.12). В распоряжении разработчика также предоставляются новые инструменты, предназначенные для оптимизации приложений, рассчитанных на работу с большими экранами (табл. 1.13).

Таблица 1.12. Новые функции Android 3 (developer.android.com/sdk/android-3.0-highlights.html)

Функция	Описание
«Голографический» интерфейс пользователя	Привлекательный «трехмерный» пользовательский интерфейс
Настраиваемый главный экран	Здесь находятся виджеты, ярлыки приложений и другие объекты

[1] www.forrester.com/ER/Press/Release/0,1769,1340,00.html.

[2] www.computerworld.com/s/article/9206219/Google_Android_tablets_gain_traction_with_developers?source=CTWNLE_nlt_dailyam_2011-01-25.

Функция	Описание
Модернизированная клавиатура	Обеспечивает улучшенную точность и эффективность ввода текста
Улучшенное редактирование	Новый пользовательский интерфейс облегчает выделение, копирование и вставку текста
Панель System (Система)	Эта панель обеспечивает быстрый доступ к навигационным кнопкам, уведомлениям и статусу системы
Панель Action (Действие)	Эта панель, находящаяся в верхней части экрана, включает специфичные для приложения элементы управления (например, элементы управления навигацией)
Улучшенная многозадачность	В списке Recent Apps (Последние приложения), находящемся на панели System, отображаются задания, которые выполняются одновременно. Этот список позволяет переключаться между выполняющимися приложениями
Параметры подключений	Подключение устройства Android к клавиатуре с помощью порта USB или подключения Bluetooth
Поддержка протоколов Photo Transfer Protocol (PTP, Протокол передачи фотографий) и Media Transfer Protocol (MTP, Протокол передачи медиа)	Эти протоколы, разработанные компанией Microsoft, обеспечивают передачу файлов фотографий, видео и музыки с устройства Android на компьютер. Можно разработать приложения, с помощью которых пользователи могут создавать и управлять медифайлами, а также пересылать их на различные устройства
Модем Bluetooth	Подключение к сети Wi-Fi или 3G компьютера или других устройств с помощью модема, в качестве которого используются Android-устройства
Приложение Browser	Вместо несколько отдельных окон новое приложение Browser (Браузер) создает вкладки, облегчает просмотр «не мобильных» веб-сайтов (с помощью улучшенного масштабирования, прокрутки и других дружественных пользователю методик). Также поддерживается режим «инкогнито» при анонимном просмотре сайтов, обеспечивается поддержка мультисенсорного экрана для JavaScript и подключаемых модулей и другие подобные методики. Можно также автоматически входит на веб-сайты Google и синхронизировать закладки с Google Chrome
Приложение Camera	Это приложение переработано с учетом использования устройств с большим экраном. С помощью этого приложения пользователи могут легко получать доступ к функциям камеры, например к фронтальной камере, к вспышке, автофокусу и другим элементам управления камерой устройства Android. Функции изменения скорости видеозаписи позволяют снимать видео с меньшей частотой кадров, чем стандартная. Воспроизведение такого видео с обычной скоростью создает иллюзию ускоренного развития событий

продолжение ✍

Таблица 1.12 *(продолжение)*

Функция	Описание
Приложение Contacts (Контакты)	Пользовательский интерфейс, состоящий из двух панелей, облегчает чтение, редактирование и организацию контактов. Благодаря быстрой прокрутке облегчается быстрый поиск контактов
Приложение Email (E_mail)	Воспользуйтесь панелью Action (Действие) для распределения электронной почты по папкам и синхронизации вложений. Воспользуйтесь виджетом e-mail, находящимся на главном экране, для упрощенного просмотра сообщений
Приложение Gallery (Галерея)	Просмотр альбомов в полноэкранном режиме, на котором отображаются миниатюры, облегчающие просмотр фотографий в альбоме

Таблица 1.13. Новые функции Android 3, предназначенные для разработчиков (developer.android.com/sdk/android-3.0-highlights.html)

Функция	Описание
Обратная совместимость	Версия Android 3.x совестима с приложениями, разработанными с помощью предыдущих версий Android
«Объемный» интерфейс пользователя (Holographic UI)	Новые и разработанные ранее приложения в среде Android 3 приобретают «объемный» вид благодаря добавлению соответствующего атрибута в файл манифеста приложения
Добавление компоновки для устройств с большим экраном в существующие приложение	Добавление новых компоновок и макетов, предназначенных для устройств с большим экраном, в существующие приложения, разработанные для устройств с маленьким экраном
Activity fragments (Фрагменты действий)	Распределение действий приложений по модульным фрагментам, которые могут использоваться в различных комбинациях. В Google имеются расширения этой библиотеки API, поэтому она может использоваться в Android 1.6 и более старших версиях
Новые и обновленные виджеты интерфейса пользователя и экрана Home	В число этих виджетов входят поле поиска, календарь, 3D стек, указатель даты/времени и ряд других. Виджеты, находящиеся на экране Home, могут контролироваться с помощью касания. Эти жесты применяются для прокрутки и смахивания содержимого экрана
Action Bar	Каждое приложение имеет свою собственную постоянную панель Action, предлагающую пользователям возможности навигации и ряд других возможностей
Расширения для игр	К числу расширений для игр относятся следующие: • Улучшенная производительность, обеспечиваемая за счет параллельного «сбора мусора», более быстрого распределения событий и улучшенным видеодрайверам.

Функция	Описание
	• Обработка событий, связанных с вводом собственных данных и сенсорами. • Новые сенсоры (гироскоп, барометр, сенсор гравитации и ряд других) обеспечивают улучшенную обработку 3D-перемещений. • Библиотека API Khronos OpenSL ES API предназначена для воспроизведения собственного аудио. • Библиотека Khronos EGL предназначена для управления собственной графикой. • Собственный доступ к Activity Lifecycle (Жизненный цикл действия) и к API, предназначенным для управления окнами. • Библиотеки API Native Asset Manager (Диспетчер собственных ресурсов) и Storage Manager (Диспетчер хранилища).
Дополнительные возможности уведомлений	С помощью класса-построителя (builder) выполняется добавление в уведомления приложений больших и маленьких значков, заголовков и флагов приоритета
Буфер обмена	С помощью буфера обмена пользователи могут копировать и вставлять данные в различные приложения
Перетаскивание	Фреймворк `DragEvent` применяется для добавления функций перетаскивания в приложение
Множественное выделение	С помощью этой функции пользователи могут выделять несколько элементов в списке или таблице
Протокол Media/Picture Transfer Protocol (MTP/PTP, Передача медиафайлов/изображений)	Благодаря этой функции пользователи могут легко передавать любые типы медиафайлов с устройств Android на главные компьютеры
Поддержка архитектуры процессором с несколькими ядрами	Обеспечивает выполнение Android 3.x на архитектурах с одним ядром или несколькими ядрами (для улучшения производительности)
Технология http Live Streaming (HLS)	Приложения могут поддерживать URL-ссылки для мультимедийных списков воспроизведения в медиафреймворке, используемые для открытия сеанса HTTP Live Streaming. В результате обеспечивается высококачественная поддержка адаптивного видео
Графический механизм Renderscript 3D	Создание высокопроизводительной 3D-графики для приложений, виджетов и прочих применений, а также перенос вычислений, выполняемых при отображении графики, на графический процессор (GPU, Graphics Processing Unit)
Аппаратно ускоренная двумерная графика	Новый модуль рендеринга OpenGL улучшает производительность при выполнении простых операций с графикой
Новый анимационный фреймворк	Упрощенная анимация объектов или элементов интерфейса пользователя

продолжение ⇗

Таблица 1.13 *(продолжение)*

Функция	Описание
Bluetooth A2DP и HSP	Приложения API для профилей Bluetooth Advanced Audio Distribution Profile (A2DP) и Headset Profile (HSP) обеспечивают проверку наличия подключенных устройств Bluetooth, уровень заряда аккумуляторной батареи и некоторые другие параметры
Фреймворк Digital Rights Management (DRM, Управление цифровыми правами)	Библиотека API, которая позволяет управлять защищенным контентом в приложениях
Новые политики для приложений, выполняющих управление устройствами	Корпоративные приложения по администрированию устройств, поддерживающие различные политики, такие как истечение срока действия паролей и ряд других

1.6. Android Ice Cream Sandwich

Версия Android Ice Cream Sandwich («сэндвич с мороженым») появилась в октябре 2012 года. Она сочетает в себе возможности Android 2.3 (Gingerbread) и Android 3.0 (Honeycomb) и предназначена для использования на всех устройствах Android. Эта версия Android позволяет включить функции, поддерживаемые в версии Honeycomb («голографический» интерфейс пользователя, новый загрузчик и ряд других функций, ранее доступных только для планшетов), в приложения, предназначенные для смартфонов. В результате обеспечивается простое масштабирование приложений, позволяющее их использовать на различных устройствах В версии Ice Cream Sandwich также появился ряд новых функцийу (табл. 1.14).

Таблица 1.14. Некоторые функции Android Ice Cream

Функция	Описание
0-click NFC Peer-to-Peer Sharing (Общий одноранговый доступ к NFC, не требующий вмешательства пользователя)	Пользователи совместимых Android-устройств могут получать общий доступ к контенту (например, к контактам и видео) путем размещения устройств поблизости друг от друга
Head tracking (Отслеживание лиц)	С помощью камеры совместимые устройства могут определять положение глаз, носа и рта пользователя. Камера может также отслеживать направление взгляда пользователя, позволяя создавать приложения, которые изменяют перспективу на основе взгляда пользователя (перерисовывание ландшафтов в трехмерных играх)
Virtual camera operator (Виртуальный оператор камеры)	В процессе видеосъемки камера автоматически фокусируется на говорящем человеке. Например, если два человека относятся к одной стороне видеочата, камера идентифицирует говорящего человека из этой пары и сфокусируется на нем

Функция	Описание
Android@Home framework (Фреймворк Android@ Home)	Обеспечивает создание приложений Android, контролирующих находящиеся у пользователя дома бытовые приборы, например выключатели освещения (которые обычно контролируют специальные лампы), настройку термостатов, контроль системы полива и выполнение других подобных задач

1.7. Загрузка приложений из Android Market

На время выхода этой книги в Google Android Market были доступны сотни тысяч приложений, и это число постоянно росло. В табл. 1.15 перечислены некоторые популярные приложения Android. Приложения можно загрузить из Android Market непосредственно на ваше устройство Android. Также Android Market рассылает уведомления о наличии обновлений для загруженных приложений.

Таблица 1.15. Некоторые популярные приложения Android на Android Market

Категория Android Market	Примеры приложений
Comics (Комиксы)	Marvel Superheroes, Dilbert Calendar, Jerry Seinfeld Jokes
Communication (Связь)	Google Voice, Skype mobile™, Wi-Fi Locator, Easy
Entertainment (Развлечения)	Face Melter, Fingerprint Scanner, Fandango® Movies
Finance (Финансы)	Mint.com Personal Finance, PayPal, Debt Payoff Planner
Games: Arcade & Action (Игры: аркады и экшны)	NESoid, Droid Breakout, Raging Thunder 2 Lite, Whac 'em!
Games: Brain & Puzzle (Игры: головоломки)	Enjoy Sudoku, Spin Cube Lite, Ultimate Simpson Puzzle
Games: Cards & Casino (Игры: азартные)	Texas Hold'em Poker, Tarot Cards, Chessmaster™
Games: Casual (Игры: логические)	City Mayor, LOL Libs, Paper Toss, SuperYatzy Free Edition
Health (Здоровье)	Fast Food Calorie Counter, CardioTrainer, StopSmoking
Lifestyle (Стиль жизни)	Zillow Real Estate, Epicurious Recipe App, Family Locator
Multimedia (Мультимедиа)	Pandora Radio, Shazam, Last.fm, iSyncr, Camera Illusion
News & Weather (Новости и погода)	The Weather Channel, CNN, NYTimes, FeedR News Reader
Productivity (Бизнес)	Adobe® Reader®, Documents To Go 2.0 Main App
Reference (Справочники)	Google Sky Map, Dictionary.com, Wikidroid for Wikipedia
Shopping (Покупки)	Gluten Free, Amazon.com, Barcode Scanner, Pkt Auctions eBay

продолжение ⮫

Таблица 1.15 *(продолжение)*

Категория Android Market	Примеры приложений
Social (Социальные сети)	Facebook®, Twitter for Android, MySpace, Bump, AIM
Sports (Спорт)	NFL Mobile, Nascar Mobile, Google Scoreboard
Themes (Темы)	Pixel Zombies Live Wallpaper, Aquarium Live Wallpaper
Tools (Инструменты)	Compass, Droidlight LED Flashlight, AppAlarm Pro
Travel (Путешествия)	Google Earth, Yelp®, Urbanspoon, WHERE, XE Currency
Demo (Демо-версии)	Screen Crack, Bubbles, CouponMap, SnowGlobe
Software libraries (Библиотеки программ)	Translate Tool, Security Guarder, Car Locator Bluetooth Plugin

Посетите веб-сайт market.android.com и найдите требуемые приложения либо посетите сайты с рекомендациями по выбору и использованию приложений (абл. 1.16). Здесь вы найдете как бесплатные, так и платные приложения. Цены на приложения, распространяемые через Android Market, устанавливают разработчики, получая 70 % от прибыли. Многие разработчики приложений используют маркетинговую стратегию, суть которой заключается в том, что базовая версия приложения распространяется бесплатно. Если же пользователю понравилась эта версия, он может приобрести полную версию приложения, обладающую расширенным набором функций. Эта так называемая «облегченная» стратегия подробно рассматривается в разделе 2.10 главы 2.

Таблица 1.16. Обзор приложений Android и веб-сайтов с рекомендациями

Название	URL-ссылка
AppBrain	www.appbrain.com/
AndroidLib	www.androlib.com/
Android Tapp™	www.androidtapp.com/
Appolicious™	www.androidapps.com/
AndroidZoom	www.androidzoom.com/
doubleTwist®	www.doubletwist.com/apps/
mplayit™	mplayit.com/#homepage

1.8. Пакеты

В Android используется целая коллекция пакетов, являющихся группами связанных предварительно определенных классов. Некоторые из пакетов специфичны для Android, другие относятся к Java и Google. С помощью пакетов обеспечивается удобный доступ к функциям операционной системы Android, а также включение этих функций в приложения. Большая часть пакетов написана на языке программирования Java и доступна из

Java-программ. С помощью пакетов Android облегчается создание Android-приложений с уникальным внешним видом и особенностями интерфейса пользователя. В табл. 1.17 перечислены пакеты, которые будут рассмотрены в книге. Полный перечень пакетов Android приводится на веб-сайте developer.android.com/reference/packages.html.

Таблица 1.17. Пакеты Android, Java и Google, используемые в книге, с указанием главы, в которой они впервые встречаются

Пакет	Описание
android.app	Включает классы высокого уровня в модели приложения Android (приложение Tip Calculator в главе 4)
android.os	Службы операционной системы (приложение Tip Calculator в главе 4)
android.text	Отображение и отслеживание текста на экране устройства (приложение Tip Calculator в главе 4)
android.widget	Классы интерфейса пользователя, предназначенные для виджетов (приложение Tip Calculator в главе 4)
android.net	Классы доступа к Сети (приложение Favorite Twitter® Searches в главе 5)
android.view	Классы интерфейса пользователя, предназначенные для взаимодействия с пользователем и компоновок (приложение Favorite Twitter® Searches в главе 5)
java.io	Потоки, сериализация и доступ к файловой системе для устройств ввода и вывода (приложение Flag Quiz в главе 6)
java.util	Классы утилиты (приложение Favorite Twitter® Searches в главе 5)
android.content.res	Классы, предназначенные для обеспечения доступа к ресурсам приложения (например, к медиафайлам, цветам, рисункам и прочим ресурсам), а также к информации о конфигурации устройства, влияющей на поведение приложения (приложение Flag Quiz Game в главе 6)
android.graphics.drawable	Классы, предназначенные только для отображаемых на экране элементов (например, градиентов) (приложение Flag Quiz Game в главе 6)
android.media	Классы, предназначенные для обработки медиаинтерфейсов аудио и видео (приложение Spotz Game в главе 8)
android.util	Методы утилит и утилиты XML (приложение Cannon Game в главе 7)
android.content	Доступ и публикация данных на устройстве (приложение Doodlz в главе 9)
android.hardware	Поддержка аппаратного обеспечения устройств (приложение Doodlz в главе 9 и приложение Enhanced Slideshow в главе 13)
android.provider	Доступ к провайдерам контента Android (приложение Doodlz в главе 9)
android.database	Обработка данных, возвращаемых провайдером контента (приложение Address Book в главе 10)
android.database.sqlite	Управления базами данных SQLite для частных баз данных (приложение Address Book в главе 10)

продолжение ⇗

Таблица 1.17 *(продолжение)*

Пакет	Описание
android.graphics	Графические инструменты, применяемые для рисования на экране (приложение Route Tracker в главе 11)
android.location	Службы, основанные на локации (приложение Route Tracker в главе 11)
com.google. android. maps	Используется в приложении Route Tracker, глава 11
android.appwidget	Используется в приложении Weather Viewer, глава 14
java.net	Сетевые классы (например, обработка адресов Интернета и запросы HTTP) (приложение Weather Viewer в главе 14)
javax.xml.parsers	Обработка XML-документов (приложение Weather Viewer в главе 14)
org.xml.sax	Simple API для XML (SAX API), используемые для чтения данных из XML-документов (приложение Weather Viewer в главе 14)

1.9. Android Software Development Kit (SDK)

В состав Android SDK включены инструменты, предназначенные для создания Android-приложений. Этот набор инструментов доступен на веб-сайте Android Developers (на бесплатной основе) (см. раздел «Подготовительные действия» в предыдущей главе, где описан порядок загрузки из Интернета всех инструментов, применяемых для разработки Android-приложений). В число этих инструментов входят Java SE, интегрированная среда разработки Eclipse, Android SDK 3.x и подключаемый модуль ADT для Eclipse.

Интегрированная среда разработки Eclipse

Интегрированная среда разработки Eclipse рекомендуется к использованию при разработке Android-приложений. Несмотря на эти рекомендации, для создания Android-приложений можно использовать текстовый редактор и инструменты командной строки. В среде Eclipse поддерживается множество языков программирования, в том числе Java, C++, C, Python, Perl, Ruby на Rails и ряд других. Подавляющее большинство приложений Android создаются на языке Java. В состав Eclipse входят следующие компоненты и функции:

- редактор кода с поддержкой выделения синтаксиса цветом и нумерации строк;
- автоматические отступы и автозавершение (то есть подсказки при вводе кода);
- отладчик;
- система контроля версий;
- поддержка рефакторинга.

Среда Eclipse применяется в разделе 1.11 для тестирования приложения Doodlz. Начиная с главы 3, где разрабатывается приложение **Welcome**, среда Eclipse применяется для разработки приложений.

Плагин Android Development Tools (ADT) для Eclipse

Плагин ADT (Android Development Tools, Инструменты разработки Android-приложений) для Eclipse — это расширение интегрированной среды разработки Eclipse. С помощью этого подключаемого модуля можно создавать, выполнять и отлаживать приложения Android, экспортировать их для дальнейшего распространения (например, выгружать на Android Market) и выполнять ряд других операций. Плагин ADT также включает визуальный инструмент, применяемый для создания графического интерфейса пользователя. Чтобы спроектировать интерфейс с помощью этого инструмента, достаточно воспользоваться готовыми компонентами, которые компонуются путем обычного перетаскивания. Более подробно плагин ADT рассматривается в главе 3, где он применяется для разработки приложения Welcome.

Эмулятор Android

Эмулятор Android, включенный в состав Android SDK, позволяет создать имитационную среду для запуска приложений Android под управлением Windows, Mac OS X либо Linux. Эмулятор отображает весьма реалистичное окно интерфейса пользователя Android. Перед запуском приложения на выполнение следует создать *AVD (Android Virtual Device, Виртуальное устройство Android)*. Это устройство определяет функциональные характеристики реального устройство Android, на котором нужно тестировать приложения. А именно, аппаратное обеспечение, системный образ, размер экрана, хранилище данных и ряд других характеристик. Если нужно тестировать приложения на нескольких устройствах Android devices, придется создать отдельные AVD, имитирующие уникальные физические устройства.

Большинство экранных снимков в книге было сделано именно с помощью эмулятора, а не реального устройства Android. С помощью эмулятора можно сымитировать большинство жестов Android (табл. 1.18) и элементов управления (табл. 1.19), используя клавиатуру и мышь компьютера. Конечно, возможности воспроизведения жестов с помощью эмулятора несколько ограничены, поскольку компьютер не в состоянии сымитировать все функции аппаратуры Android. Например, чтобы протестировать GPS-приложения с помощью эмулятора, нужно создать файлы, имитирующие данные GPS. Хотя можно смоделировать изменения ориентации (переключение в портретный/альбомный режим), не существует способа смоделировать показания *акселерометра* (это устройство оценивает ориентацию и наклон устройства). Чтобы протестировать эти и подобные им функции, загрузите приложение на физическое устройство Android. Дополнительные сведения по этой теме приведены в главе 11, где создается приложение Route Tracker. В главе 3 будет создано устройство AVD и использован эмулятор для создания приложения Welcome.

Таблица 1.18. Имитация жестов Android на эмуляторе (developer.android.com/guide/developing/tools/emulator.html)

Жест	Действие в эмуляторе
Касание	Щелкните левой кнопкой мыши (приложение Tip Calculator в главе 4)
Двойное касание	Дважды щелкните левой кнопкой мыши (приложение Cannon Game в главе 7)
Длинное нажатие	Щелкните левой кнопкой мыши и удерживайте ее
Перетаскивание	Щелкните левой кнопкой мыши, удерживайте ее и перетаскивайте мышь (приложение Cannon Game в главе 7)
Пощелкивание	Щелкните левой кнопкой мыши и, не отпуская ее, переместите указатель в направлении пощелкивания (приложение Address Book в главе 10)
Смахивание	Щелкните левой кнопкой мыши и, удерживая ее, переместите указатель мыши в направлении смахивания, а потом быстро отпустите кнопку мыши (приложение Address Book в главе 10)
Масштабирование двумя пальцами	Нажмите и удерживайте клавишу Ctrl. Появятся две окружности, имитирующие касание экрана двумя пальцами. Перемесите окружности в начальную позицию, щелкните левой кнопкой мыши и, удерживая ее, переместите окружности в конечную позицию (приложение Route Tracker в главе 11)

Таблица 1.19. Имитация органов управления Android с помощью эмулятора (имитация дополнительных органов управления описана на сайте developer.android.com/guide/developing/tools/emulator.html)

Орган управления устройства Android	Действие в эмуляторе
Пиктограмма Back (Назад)	Esc
Вызов/набор номера	F3
Камера	Ctrl+5 на цифровой клавиатуре, Ctrl+F3
Завершить вызов	F4
Пиктограмма Home (Домой)	Home
Меню (левая программная кнопка)	F2 или Page Up
Питание	F7

Орган управления устройства Android	Действие в эмуляторе
Поиск	F5
* (правая программная кнопка)	Shift+F2 или PageDown
Вращение влево	7 на цифровой клавиатуре, Ctrl+F11
Вращение вправо	9 на цифровой клавиатуре, Ctrl+F12
Включение/выключение сети мобильной связи	F8
Кнопка увеличения громкости	+ на цифровой клавиатуре, Ctrl+F5
Кнопка уменьшения громкости	- на цифровой клавиатуре, Ctrl+F6

1.10. Краткий обзор объектной технологии

При разработке современных и эффективных программ достаточно трудно выполнить такие требования, как быстрота разработки, корректность и экономичность. Современные методики программирования основаны на использовании *объектов* (или более точно, *объектов классов*, представляющих собой повторно используемые программные компоненты, см. главу 3). В качестве объектов может использоваться дата, время, видео, человек, автомобиль и другие предметы материального мира. Практически каждое *существительное* может быть адекватным образом представлено программным объектом в терминах *атрибутов* (например, имя, цвет и размер) и *поведений* (например, вычисление, перемещение и коммуникация). Разработчики программ пришли к выводу, что использование модульной структуры и объектно-ориентированного проектирования при разработке приложений способствуют более продуктивной работе. Этот подход пришел на смену применявшемуся ранее структурному программированию. К тому же объектно-ориентированные программы зачастую проще понимать, корректировать и модифицировать.

Автомобиль в качестве объекта

Чтобы лучше понять суть объектов и относящегося к ним контента, воспользуемся простой аналогией. *Представьте себе, что вы находитесь за рулем автомобиля и нажимаете педаль газа, чтобы набрать скорость.* А теперь ответим на вопрос, что должно произойти до того, как вы получите возможность водить автомобиль? Прежде чем вы получите возможность водить автомобиль, его нужно изготовить. Изготовление любого автомобиля начинается с инженерных чертежей (*калек*), которые подробно описывают устройство автомобиля. На этих чертежах даже показано устройство педали акселератора. За этой педалью *скрываются* сложные механизмы, которые фактически ускоряют автомобиль подобно тому, как педаль тормоза скрывает механизмы, тормозящие автомобиль, а руль «скрывает» механизмы, поворачивающие автомобиль. Благодаря

этому люди, не имеющие понятия о внутреннем устройстве автомобиля, могут легко им управлять.

Подобно тому как невозможно готовить пищу на кухне, которая находится лишь на листе бумаги, нельзя водить автомобиль, существующий лишь в чертежах. Прежде чем вы сядете за руль машины, ее нужно воплотить в металл на основе инженерных чертежей. Воплощенный в металле автомобиль имеет *реальную* педаль газа, с помощью которой он может ускоряться, но, к счастью, он не может это делать самостоятельно, а только с помощью водителя.

Методы и классы

Воспользуемся примером с автомобилем для иллюстрации некоторых ключевых концепций объектно-ориентированного программирования. Для выполнения задачи в программе требуется *метод*, «скрывающийся» в инструкциях программы, которые фактически выполняют задание. Метод скрывает эти инструкции от пользователя подобно тому, как педаль газа автомобиля скрывает от водителя механизмы, вызывающие ускорение автомобиля. Программная единица, именуемая *классом*, включает методы, выполняющие задачи класса. Например, класс, представляющий банковский счет, может включать три метода, один из которых выполняет *пополнение счета*, второй — *вывод средств* со счета, а третий — *запрос текущего баланса*. Класс подобен концепции автомобиля, представленного инженерными чертежами, которые также включают чертеж педали газа, рулевого колеса и других механизмов.

Создание экземпляра класса

Подобно тому как *изготовление автомобиля* на основе чертежей — обязательное условие его вождения, *создание объекта* класса — обязательное условие выполнения задач, реализованное на основе методов этого класса. Этот процесс называется *созданием экземпляра*. Полученный при этом объект называется *экземпляром* класса.

Повторное использование

Подобно тому, как на основе одних и тех же чертежей *можно создать много* автомобилей, на основе одного класса *можно создать много* объектов. Благодаря использованию существующих классов для создания новых классов экономится время и силы разработчика. Повторное использования также облегчает создание более надежных и эффективных систем, поскольку ранее созданные классы и компоненты не нуждаются в *тестировании*, *отладке* и *оптимизации производительности*. Подобно тому, как концепция использования *взаимозаменяемых* частей легла в основу индустриальной революции, повторно используемые классы — двигатель прогресса в области создания программ, который был вызван внедрением объектной технологии.

Сообщения и вызовы методов

В процессе вождения автомобиля в результате нажатия педали газа отсылается *сообщение* автомобилю, содержащее запрос на выполнение определенной задачи (ускорение автомобиля). Подобным же образом *отсылаются сообщения объекту*. Каждое сообщение

представляет собой *вызов метода*, который «сообщает» методу объекта о необходимости выполнения определенной задачи. Например, программа может вызвать метод *депозит* определенного объекта банковского счета, чтобы пополнить банковский счет.

Атрибуты и переменные экземпляра класса

Любой автомобиль помимо возможности выполнять определенные задачи также обладает *атрибутами*, такими как цвет, количество дверей, запас топлива в баке, показания спидометра и одометра. Подобно возможностям по выполнению определенных действий, атрибуты автомобиля представлены на инженерных диаграммах, представляющих собой часть проекта (в качестве атрибутов автомобиля могут выступать одометр и указатель уровня бензина). При вождении автомобиля его атрибуты перемещаются вместе с ним. Каждому автомобилю присущ *собственный* набор атрибутов. Например, каждый автомобиль «знает» о том, сколько бензина осталось в *его* баке, но ничего не «знает» о запасах горючего в баках *других* автомобилей.

Объект, как и автомобиль, имеет собственный набор атрибутов, которые он «переносит» с собой при использовании этого объекта в программах. Эти атрибуты определены в качестве части объекта класса. Например, объект bankaccount имеет *атрибут баланса*, представляющий количество средств на банковском счету. Каждый объект bankaccount «знает» о количестве средств на *собственном* счету, но ничего не «знает» о размерах *других* банковских счетов. Атрибуты определяются с помощью других переменных экземпляра класса.

Инкапсуляция

Классы *инкапсулируют* атрибуты и методы в объекты (атрибуты и методы объекта между собой тесно связаны). Объекты могут обмениваться информацией между собой, но обычно они не «знают» о деталях реализации других объектов, которые *скрыты* внутри самих объектов. Подобное *сокрытие информации* жизненно важно в практике хорошего программного инжиниринга.

Наследование

С помощью *наследования* можно быстро и просто создать новый класс объектов. При этом новый класс наследует характеристики существующего класса, которые при этом могут частично изменяться. Также в новый класс добавляются свойства, которые являются уникальными и присущими только этому классу. Если вспомнить аналогию с автомобилем, «трансформер» является объектом более обобщенного класса «автомобиль», у которого может подниматься или опускаться крыша.

Объектно-ориентированный анализ и проектирование

А теперь ответьте на вопрос, каким образом вы собираетесь программировать? Скорее всего, таким же образом, как и большинство других программистов, — включите компьютер и начнете вводить исходный код программы. Подобный подход годится при создании маленьких программ, но что делать в том случае, когда приходится создавать крупный программный комплекс, который, например, управляет тысячами

автоответчиков в колл-центре крупного банка? Либо если вам приходится возглавлять команду из 1000 программистов, занятых разработкой системы управления воздушным движением следующего поколения? Начать выполнение столь крупных и сложных проектов с того, что сесть за компьютер и вводить код, означает потерпеть неудачу.

Чтобы выработать наилучшее решение, следует выполнить процедуру детального анализа процесса определения требований к программному проекту (то есть определить *задачи*, выполняемые системой) и разработать *проект*, который будет соответствовать этим требованиям (то есть определить, каким образом система будет выполнять поставленные перед ней задачи). В идеале перед началом создания кода следует выполнить эту процедуру и тщательно проанализировать проект (либо поручить выполнение этой задачи коллегам-профессионалам). Если в ходе выполнения этого процесса происходит анализ и проектирование системы с применением объектно-ориентированного подхода, значит мы имеем дело с процессом *OOAD (object-oriented analysis and design, объектно-ориентированный анализ и проектирование)*. Языки программирования, подобные Java, называются объектно-ориентированными. Принципы объектно-ориентированного проектирования могут быть внедрены на практике с помощью подобных языков программирования.

1.11. Тестирование приложения Doodlz на виртуальном устройстве AVD

В этом разделе вы запустите на выполнение и будете «общаться» со своим первым приложением Android. С помощью приложения Doodlz пользователь может «рисовать» на экране, выбирая различные цвета и кисти разных размеров. Это приложение будет создано в главе 9. Следующая пошаговая инструкция предназначена для импортирования проекта приложения в среду Eclipse и выполнения тестирования на устройстве AVD (Android Virtual Device, Виртуальное устройство Android). Создание и настройка устройства AVD описаны в разделе «Подготовительные действия», находящемся во вводной части книги. Позднее будет рассмотрено, каким образом можно выполнять это приложение на реальном устройстве Android.

Экранные снимки, используемые в качестве иллюстраций, было созданы на компьютере, на котором установлены Windows 7, Java SE 6, Eclipse 3.6.1, Android 2.2/2.3/3.0 и подключаемый модуль ADT для Eclipse.

1. *Проверьте конфигурацию вашей системы.* Убедитесь в том, что ваш компьютер настроен в полном соответствии с требованиями, изложенными в разделе «Подготовительные действия», находящегося во вводной части книги.
2. *Откройте среду Eclipse.* Чтобы запустить Eclipse, откройте папку, в которой находится Eclipse, в вашей системе и дважды щелкните на значке Eclipse. Если Eclipse запускается первый раз, появится вкладка Welcome (Добро пожаловать), показанная на рис. 1.1. Щелкните на кнопке Workbench (Рабочая среда), находящейся на этой вкладке, чтобы закрыть вкладку и выбрать представление, используемое для разработки программ, которое носит формальное название «Java perspective in Eclipse».

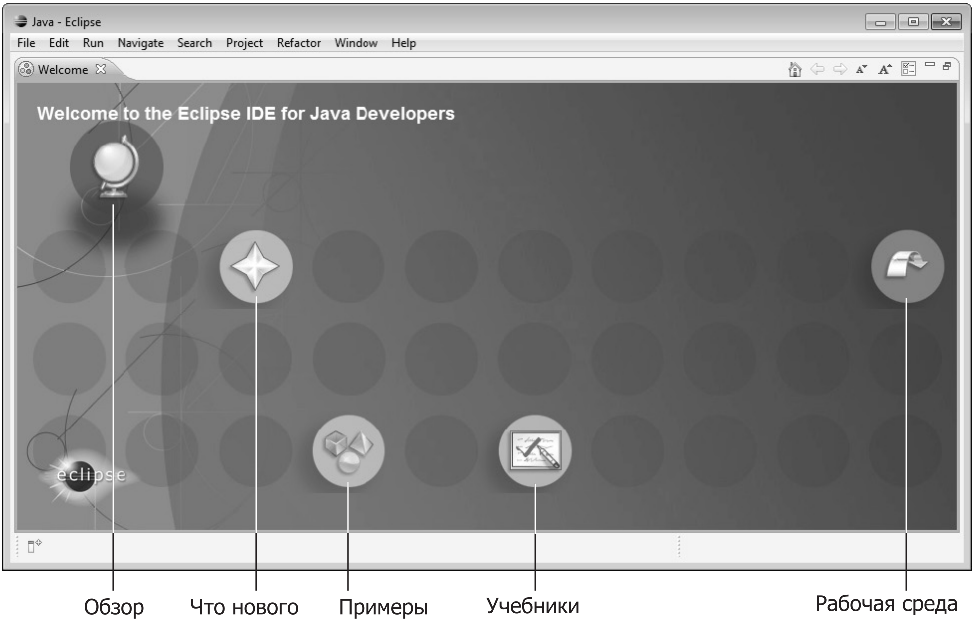


Рис. 1.1. Вкладка Welcome в окне Eclipse

3. *Откройте диалоговое окно* Import *(Импорт)*. Выполните команды File ▸ Import... (Файл ▸ Импорт), чтобы открыть диалоговое окно Import (рис. 1.2).

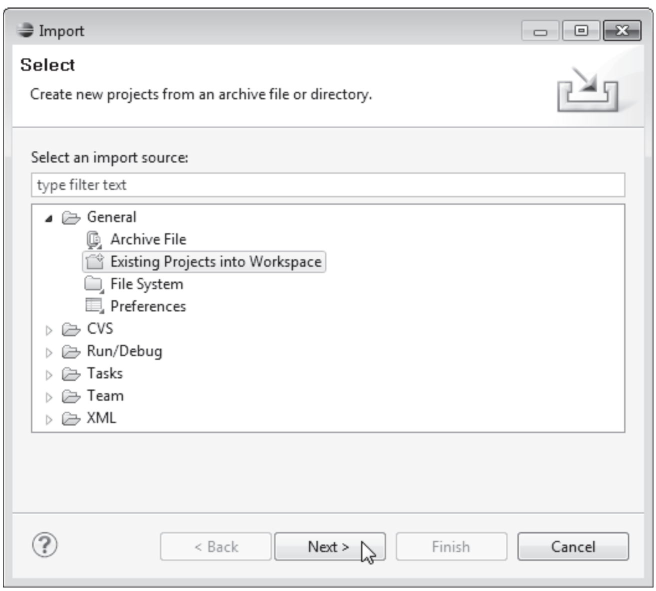


Рис. 1.2. Диалоговое окно Import

4. *Импорт проекта приложения* Doodlz. В диалоговом окне Import раскройте узел General (Общий), выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую среду), потом щелкните на кнопке Next> (Далее>) для выполнения шага Import Projects (Импорт проектов), как показано на рис. 1.3. Установите переключатель Select root directory (Выберите корневой каталог), потом щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папок), показанном на рис. 1.4, найдите папку Doodlz, которая находится в папке примеров книги, выберите ее и щелкните на кнопке OK. Нажмите кнопку Finish (Готово), чтобы импортировать проект в Eclipse. Проект появится в окне Package Explorer (Диспетчер пакетов), которое показано на рис. 1.5. Это окно находится в левой части окна Eclipse.

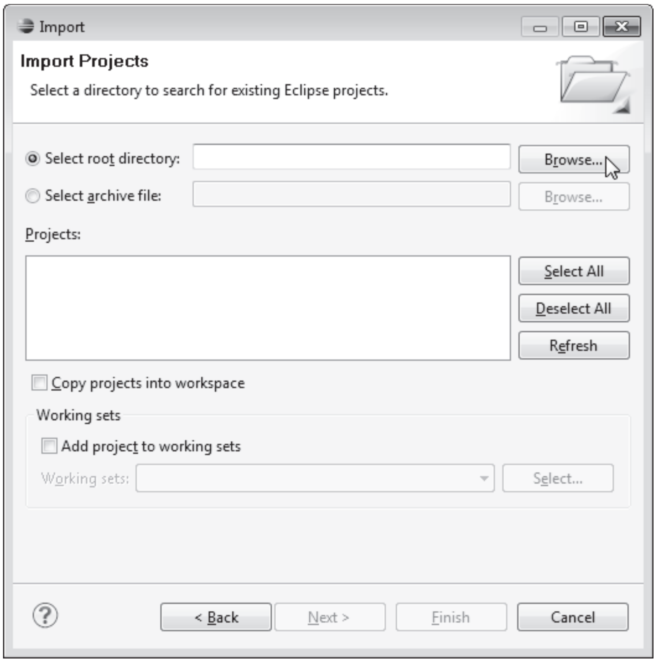


Рис. 1.3. Шаг Import Projects, выполняемый в окне Import

5. *Запуск приложения* Doodlz. В среде Eclipse выберите проект Doodlz в окне Package Explorer (см. рис. 1.5), затем выполните команду Run As ▸ Android Application (Выполнить как ▸ Приложение Android), выбрав ее в раскрывающемся меню кнопки Run As панели инструментов интегрированной среды разработки (рис. 1.6). В результате начнет выполняться приложение Doodlz на виртуальном устройстве NexusS Android Virtual Device (AVD) (рис. 1.7), создание которого описано в разделе «Подготовительные действия» вводной главы. Если нужно протестировать приложение на различных AVD-устройствах, выполните команды Window ▸ Android SDK and AVD Manager (Окно ▸ Диспетчер SDK и AVD), потом выберите нужное устройство AVD и щелкните на кнопке Start... (Запуск...). Если в момент запуска приложения уже

выполняются несколько устройств AVD, появится диалоговое окно Android Device Chooser (Выбор устройства Android), где можно выбрать устройство AVD, на котором будет выполняться приложение. Подробное рассмотрение окна Android Device Chooser будет выполнено позднее.

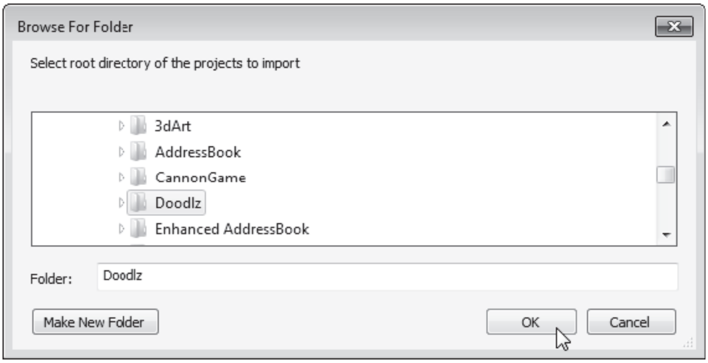


Рис. 1.4. Диалоговое окно Browser For Folder

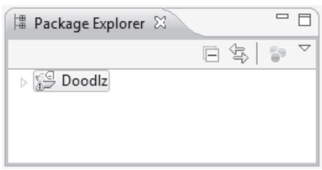


Рис. 1.5. Окно Package Explorer в среде Eclipse

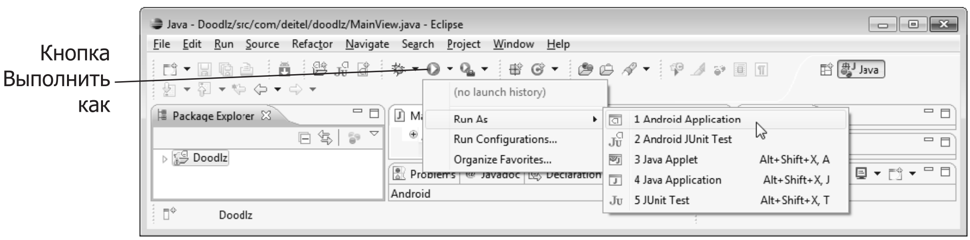


Рис. 1.6. Запуск приложения Doodlz

6. *Структура AVD*. В левой части окна AVD отображается выполняющееся приложение. В правой части окна (рис. 1.8) отображаются различные кнопки, которые имитируют физические и программные кнопки реального устройства Android, и клавиатура, имитирующая физическую и программную клавиатуры устройства. Физические кнопки — это реальные кнопки устройства. Программные кнопки отображаются на сенсорном экране устройства. С помощью кнопок устройства AVD можно взаимодействовать с приложениями и операционной системой Android, выполняемыми на AVD. После установки приложения на устройстве Android можно

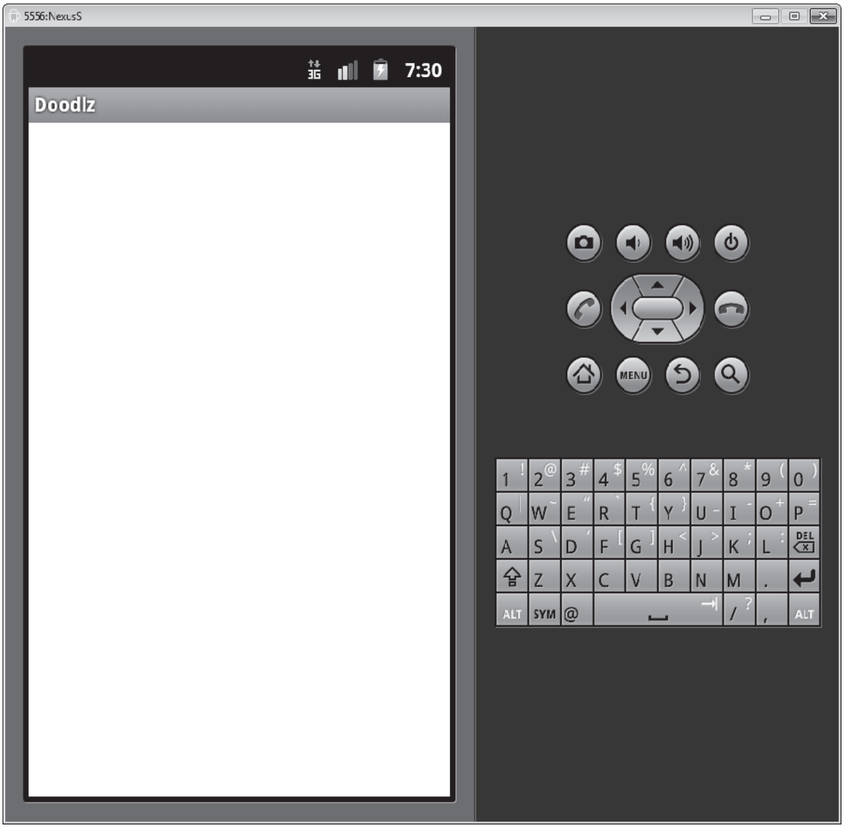


Рис. 1.7. Выполнение приложения Doodlz на виртуальном устройстве Android

приступать к созданию нового рисунка путем перетаскивания пальцем в области холста. При использовании устройства AVD «касание» поверхности экрана имитируется мышью.

7. *Отображение параметров приложения*. Чтобы отобразить параметры приложения, коснитесь кнопки Меню (Menu). Обратите внимание, что на некоторых реальных устройствах эта кнопка обозначена пиктограммой в виде горизонтальных полосок. После этого отображается меню параметров приложения (рис. 1.9). Это меню включает следующие параметры: Color (Цвет), Line Width (Ширина линии), Erase (Удалить), Clear (Очистить) и Save Image (Сохранить изображение). Выберите параметр Color, и на экране появится графический интерфейс пользователя, с помощью которого можно изменить цвет линии. Если же выбрать параметр Line Width, у вас появится возможность изменить толщину линии. Выберите параметр Erase для окрашивания рисуемых линий в белый цвет (в результате цвет полностью исчезает). После выбора параметра Clear рисунок исчезает. Если же выбрать параметр Save Image, изображение будет сохранено в галерее изображений устройства. Все эти параметры будут подробнее рассмотрены далее.

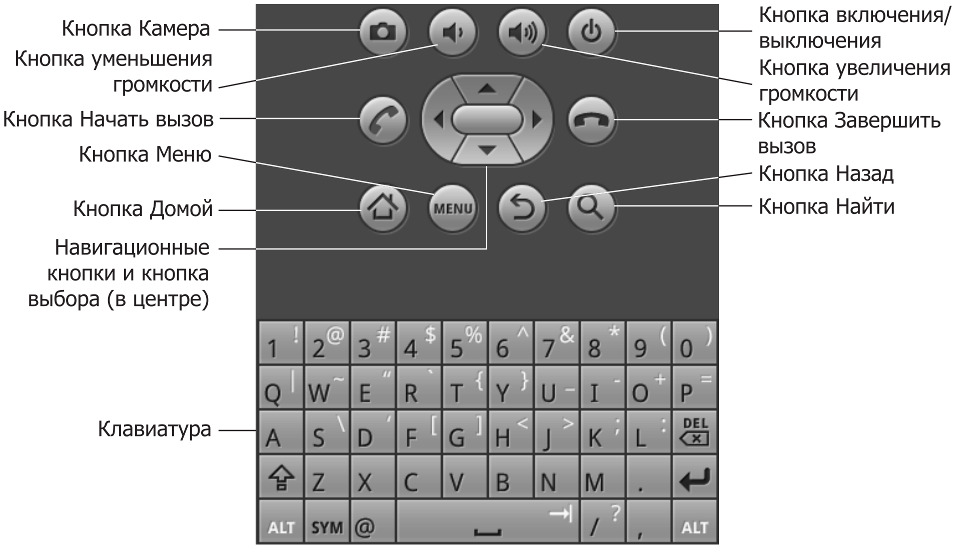


Рис. 1.8. Виртуальное устройство AVD, на котором выполняется приложение Doodlz

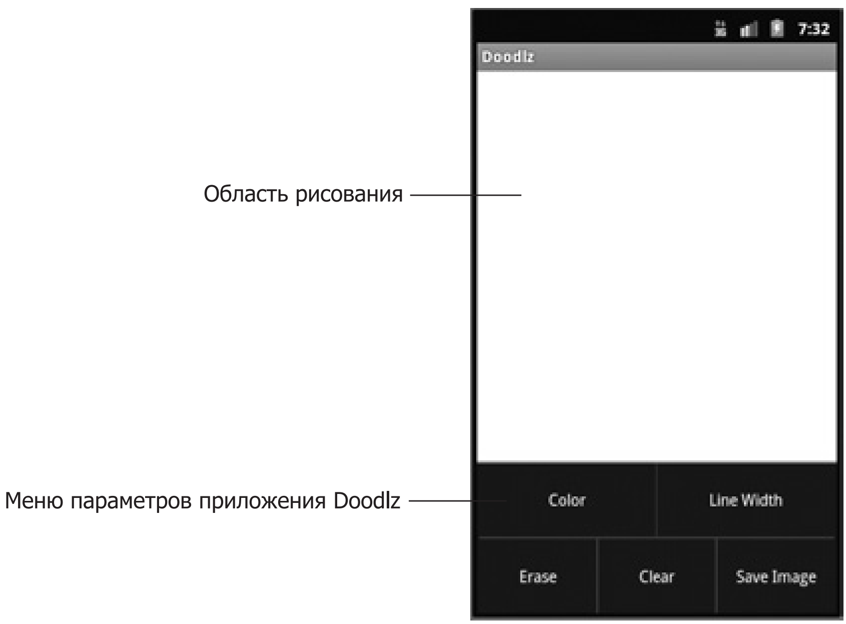


Рис. 1.9. Меню параметров приложения Doodlz

8. *Изменение цвета кисти на красный.* Чтобы изменить цвет кисти, сначала коснитесь параметра Color для отображения панели изменения цвета (рис. 1.10, *а*). Цвета определяются с помощью цветовой схемы RGBA, в которой с помощью целочисленных цветовых значений (от 0 до 255) определены следующие цветовые компоненты:

альфа, красный, зеленый и синий. Панель настройки цвета состоит из полос SeekBar Red (Красный), Green (Зеленый), Blue (Синий) и Alpha (Альфа). С помощью этих элементов направления можно выбирать интенсивность красного, зеленого и синего цветов, а также настраивать прозрачность цвета. Чтобы изменить интенсивность цвета, перетащите полосу SeekBar. В результате в окне приложения отобразится новый цвет. Чтобы выбрать красный цвет, перетащите полосу SeekBar Red в крайнее правое положение (рис. 1.10, *а*). Нажмите кнопку Done (Готово), чтобы вернуться в область рисования. Перетащите «палец» (роль которого играет мышь) по экрану и нарисуйте лепестки цветка (рис. 1.10, *б*).

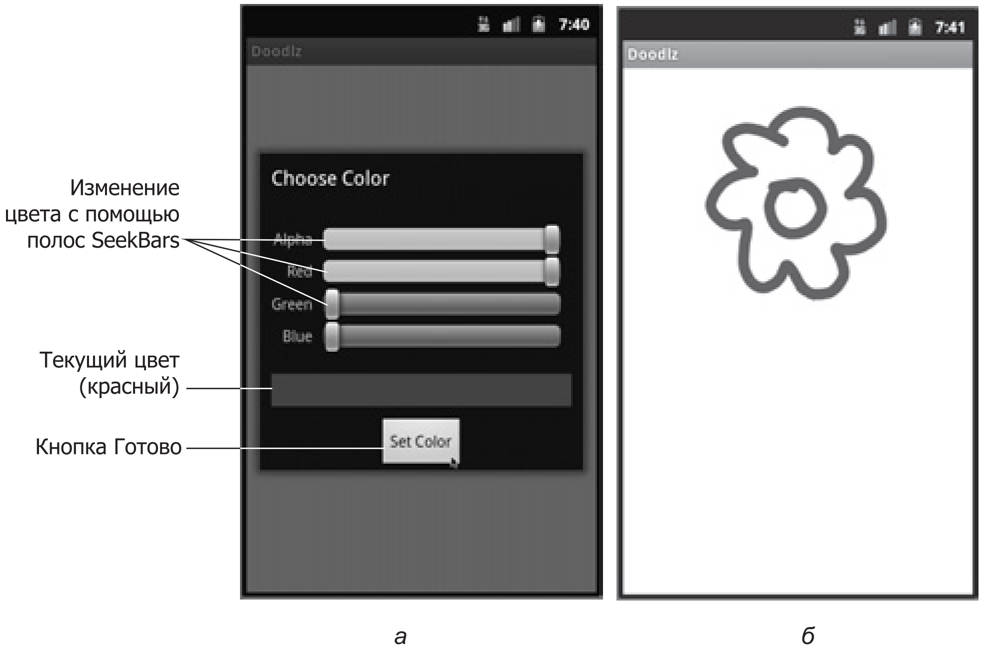


Рис. 1.10. Выбираем красный цвет кисти и рисуем лепестки цветка: *а* — выбор красного цвета кисти; *б* — рисование красных лепестков цветка

9. *Изменение цвета кисти на темно-зеленый.* Снова измените цвет кисти, коснувшись кнопки Меню с последующим выбором параметра Color. Выберите темно-зеленый цвет путем перетаскивания полосы Green SeekBar в крайнее правое положение, а полос Red и Blue SeekBars — в крайнее левое положение (рис. 1.11, *а*).
10. *Изменение толщины линии.* Чтобы изменить толщину линии, коснитесь кнопки Menu, а потом выберите параметр Line Width. Перетащите полосу SeekBar, предназначенную для настройки толщины линии, вправо (рис. 1.11, *б*). Нажмите кнопку Done, чтобы вернуться в область рисования. Нарисуйте стебель и листья цветка. Повторите шаги 9 и 10, чтобы выбрать более светлый зеленый цвет и более тонкую линию, затем нарисуйте траву (рис. 1.12).



а *б*

Рис. 1.11. Изменение цвета и толщины линии: *а* — выбор зеленого цвета для рисования; *б* — выбор более толстой линии

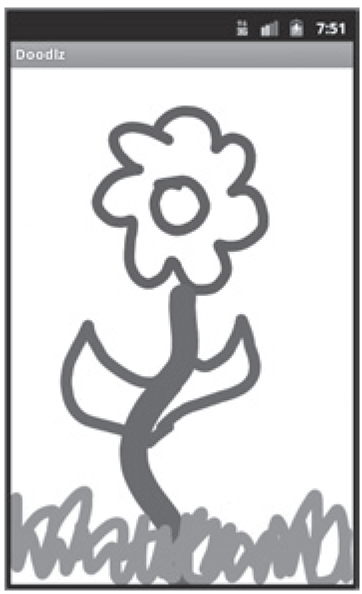
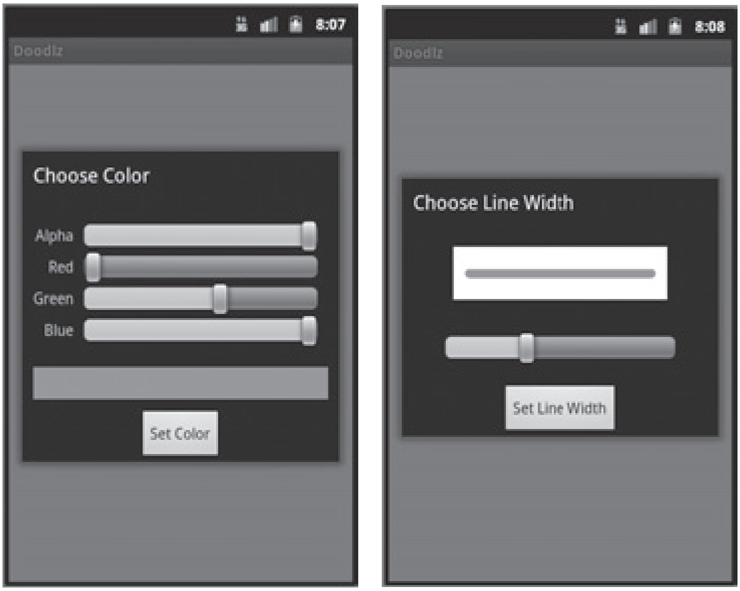


Рис. 1.12. Рисуем стебель и листья после выбора нового цвета и ширины линии

11. *Завершение рисунка*. Воспользуйтесь шагами 9–10, чтобы выбрать синий цвет линии (рис. 1.13, *а*) и сузить ее (рис. 1.13, *б*). Вернитесь к области рисования и нарисуйте капли дождя (рис. 1.14).



а *б*

Рис. 1.13. Изменение цвета и толщины линии: *а* — выбор синего цвета для рисования; *б* — выбор более тонкой линии

12. *Сохраните изображение*. При желании можно сохранить изображение в приложении Gallery (Галерея). Для этого нажмите кнопку Menu (Меню), потом нажмите кнопку Save Image (Сохранить изображение). Данное изображение и другие изображения, хранящиеся в памяти устройства, можно открыть с помощью приложения Gallery.
13. *Вернитесь на главный экран*. Вернитесь на главный экран AVD, щелкнув на кнопке Home (Домой), находящейся на экране AVD.

Выполнение приложения Doodlz на устройстве Android

Если у вас имеется устройство Android, протестируйте приложение на этом устройстве.

1. Во-первых, нужно перейти в режим отладки устройства. Запустите приложение Settings (Настройки), установленное на устройстве, затем выберите команды Applications ▸ Development (Приложения ▸ Разработка) и убедитесь в том, что флажок USB debugging (Отладка USB) установлен.

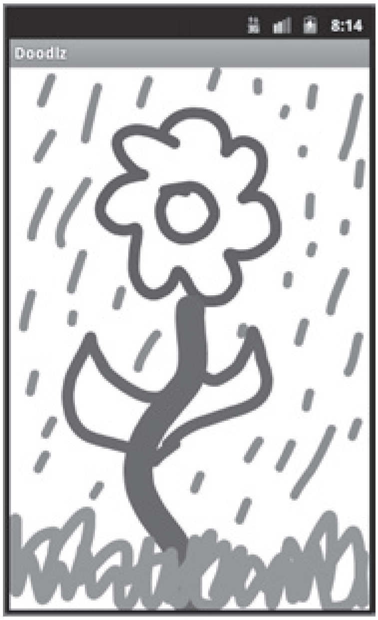


Рис. 1.14. Рисуем капли дождя после выбора нового цвета и толщины линии

2. Подключите устройство к компьютеру с помощью кабеля USB (обычно подобный кабель входит в комплект поставки устройства или же его можно приобрести отдельно).
3. В среде Eclipse перейдите в окно Package Explorer, выберите проект Doodlz, потом выполните команды Run As ▸ Android Application, выбранные в раскрывающемся меню кнопки Run As панели инструментов интегрированной среды разработки (рис. 1.6).

Если отсутствует открытое устройство AVD, но к компьютеру подключено устройство Android, Eclipse автоматически установить приложение на устройство и запустит это приложение. Если открыто одно или большее число устройств AVD и/либо к компьютеру подключены устройства Android, появится диалоговое окно Android Device Chooser (см. рис. 1.15), в котором можно выбрать виртуальное устройство AVD либо реальное устройство, на которое будет установлено и выполнено приложение. В рассматриваемом примере запущено два виртуальных устройства AVD и подключено одно физическое устройство, поэтому приложение может выполняться на одном из трех устройств. Можно установить несколько устройств AVD, с помощью которых эмулируются реальные устройства Android с различными размерами экрана и на которых выполняются различные версии операционной системы Android.

В разделе активных устройств Android (рис. 1.15) доступно лишь одно физическое устройство, подключенное к компьютеру (находится на второй позиции в списке устройств), и три виртуальных устройства AVD. Каждое устройство AVD имеет свое имя AVD, которое может выбрать пользователь (NexusS и MotorolaXoom, как показано на рисунке). Выберите используемое реальное устройство или устройство AVD, щелкните на кнопке OK для его установки и выполнения на реальном устройстве или

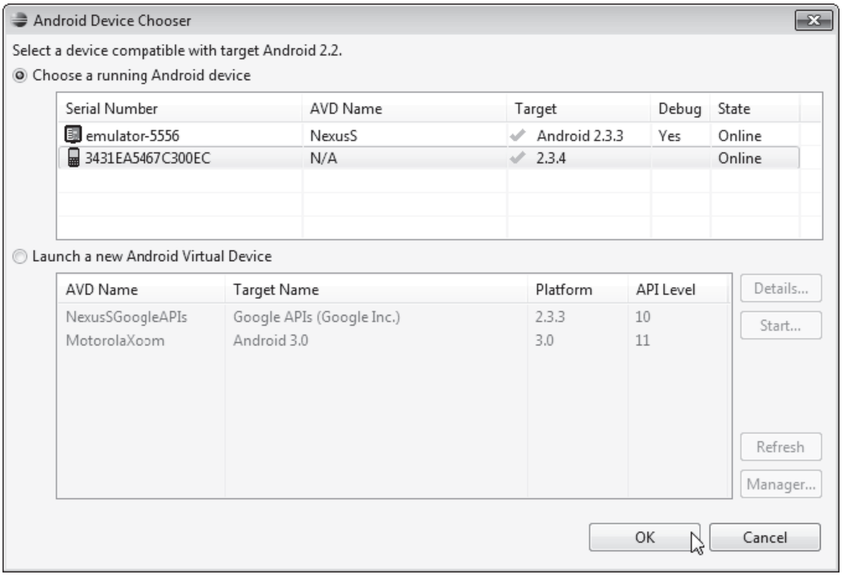


Рис. 1.15. Диалоговое окно Android Device Chooser

виртуальном устройстве AVD. Если определены другие виртуальные устройства AVD, которые в настоящее время не выполняются, выберите и вызовите на выполнение одно из устройство AVD, находящееся в нижней части диалогового окна Android Device Chooser.

Если вы создаете приложения, предназначенные для распространения через Android Market, протестируйте их на возможно большем числе реальных устройств. Не забывайте о том, что некоторые функции могут быть протестированы лишь на реальных устройствах. Если вы не можете найти реальные устройства в достаточном количестве, создайте устройства AVD, имитирующие различные устройства, на которых должно выполняться созданное вами приложение. Просмотрите в Интернете спецификации реальных устройств, а потом сконфигурируйте соответствующим образом каждое виртуальное устройство AVD. Также можно изменить файл config.ini виртуального устройства AVD, как описано в разделе «Setting hardware emulation options» на сайте developer.android.com/guide/developing/tools/avd.html.

Файл config.ini включает параметры, которые не настраиваются с помощью подключаемого модуля ADT в Eclipse. В результате изменения этих параметров можно добиться более точного соответствия выбранного виртуального устройства Android с реальным.

1.12. Ресурсы Deitel

На нашем веб-сайте (www.deitel.com) представлена информация о более чем 100 центрах ресурсов (Resource Center), содержащих справочные сведения по самым различным темам, включая языки программирования, разработку программ, Web 2.0, бизнес в Интернете и проекты с открытым исходным кодом. Центры ресурсов являются результатом

исследований, проводившихся в рамках поддержки наших публикаций, и представляют собой наш небольшой семейный бизнес. Многие замечательные ресурсы мы нашли в Интернете, в том числе справочники, документацию, программы, статьи, блоги, подкасты, видеофайлы, примеры кода, книги, электронные книги и ряд других ресурсов. И что самое главное, большинство этих ресурсов являются совершенно бесплатными. Новейшие центры ресурсов анонсируются в нашем бюллетене новостей *Deitel® Buzz Online*, в Фейсбуке и Твиттере. В табл. 1.20 представлен список ресурсов Deitel, которые помогут вам начать разработку приложений Android.

Таблица 1.20. Ресурсы Deitel, предназначенные для Android-разработчиков

Название ресурса	URL-ссылка
Страница книги *Android for Programmers: An App-Driven Approach*	www.deitel.com/books/AndroidFP/
Android Resource Center	www.deitel.com/android/
Android Best Practices Resource Center	www.deitel.com/androidbestpractices/
Java Resource Center	www.deitel.com/java/
Eclipse Resource Center	www.deitel.com/Eclipse/
SQLite 3 Resource Center	www.deitel.com/SQLite3/
Домашняя страница Deitel Resource Centers	www.deitel.com/ResourceCenters.html
Deitel на Фейсбуке	www.deitel.com/DeitelFan/
Deitel в Твиттере	@deitel
Бюллетень новостей *Deitel® Buzz Online*, рассылаемый по электронной почте	www.deitel.com/newsletter/subscribe.html

1.13. Ресурсы для Android-разработчиков

В табл. 1.21 представлен список ресурсов, предназначенных для Android-разработчиков. В табл. 1.22 перечислены видеоролики Android-разработчиков, доступные на веб-сайте developer.android.com. Чтобы получить доступ к дополнительным ресурсам, посетите наши центры ресурсов Android на веб-сайте www.deitel.com/android.

Таблица 1.21. Ресурсы и советы разработчикам на платформе Android

Ресурсы Android и советы разработчикам	URL-ссылка
Канал на YouTube Android Developers	www.youtube.com/user/androiddevelopers
Примеры Android-приложений от Google	code.google.com/p/apps-for-android/
Статья O'Reilly «Ten Tips for Android Application Development»	answers.oreilly.com/topic/862-ten-tipsfor-android-application-development/

продолжение ☞

Таблица 1.21 *(продолжение)*

Ресурсы Android и советы разработчикам	URL-ссылка
Веб-сайт Bright Hub™, на котором публикуются советы по программированию на Android и пошаговые руководства	www.brighthub.com/mobile/googleandroid.aspx
Статья «10 User Experience Tips for Successful Android Apps»	www.androidtapp.com/10-user-experiencetips-for-successful-android-apps/
Советы по быстрой разработке Android-приложений	www.droidnova.com/
Учебник «Working with XML on Android: Build Java applications for mobile devices» Майкла Гэлпина (Michael Galpin), архитектурного программиста eBay	www.ibm.com/developerworks/opensource/library/x-android/index.html
Блог Android-разработчиков	android-developers.blogspot.com/
Программа Sprint Application Developers Program	developer.sprint.com/site/global/develop/mobile_platforms/android/android.jsp
Веб-сайт Android-разработчиков T-Mobile	developer website developer.t-mobile.com/site/global/resources/partner_hubs/android/p_android.jsp
Центр разработчиков приложений Android и Windows Mobile от HTC	development developer.htc.com/
Веб-сайт Android-разработчиков от Motorola	developer.motorola.com/

Таблица 1.22. Видеоролики от Android-разработчиков

Видеоролик	URL-ссылка
Androidology, Part 1 of 3: Architecture Overview	developer.android.com/videos/index.html#v=QBGfUs9mQYY
Androidology, Part 2 of 3: Application Lifecycle	developer.android.com/videos/index.html#v=fL6gSd4ugSI
Androidology, Part 3 of 3: APIs	developer.android.com/videos/index.html#v=MPukbH6D-lY
Android Developer Soapbox: Easy for Java Developers, Build Desktop Widgets	developer.android.com/videos/index.html#v=FTAxE6SIWeI
A Beginner's Guide to Android	developer.android.com/videos/index.html#v=yqCj83leYRE
The World of List View	developer.android.com/videos/index.html#v=wDBM6wVEO70

Видеоролик	**URL-ссылка**
Android UI Design Patterns	developer.android.com/videos/index.html#v=M1ZBjlCRfz0
Writing Zippy Android Apps	developer.android.com/videos/index.html#v=c4znvD-7VDA
Casting a Wide Net for All Android Devices	developer.android.com/videos/index.html#v=zNmohaZYvPw
Building Push Applications for Android	developer.android.com/videos/index.html#v=PLM4LajwDVc

1.14. Резюме

В этой главе вашему вниманию была представлена краткая история Android и функциональные свойства этой платформы. Были рассмотрены функции операционной системы Android 2.2, 2.3 и 3.0. Представлены ссылки на некоторую ключевую документацию в Интернете, группы новостей и форумы, которые позволят вам связаться с сообществом разработчиков. Мы рассмотрели Android Market и ссылки на некоторые популярные обзоры приложений и сайты с рекомендациями. Были изучены жесты Android, а также методика их выполнения на устройствах Android и эмуляторах устройств Android. Были изложены начальные сведения о пакетах Java, Android и Google, которые позволят использовать аппаратные и программные свойства для создания Android-приложений. Многие из этих пакетов будут использованы в книге. Также были рассмотрены язык программирования Java и Android SDK. Вашему вниманию был предложен краткий обзор концепций объектной технологии, в том числе классы, объекты, атрибуты и поведения. Мы протестировали приложение `Doodlz` на эмуляторе устройства Android.

В главе 2 будет рассмотрена бизнес-сторона разработки Android-приложений. Вы узнаете, каким образом можно подготовить приложения для размещения на Android Market. Вашему вниманию будут предложены советы по формированию цен и маркетингу разработанных вами приложений.

Google Play и бизнес-вопросы, связанные с разработкой приложений

2

В этой главе...

- Признаки выдающегося приложения Android.
- Указания по созданию графического интерфейса пользователя приложения.
- Регистрация на Google Play.
- Отправка приложений на Google Play.
- Ценообразование приложений и преимущества платных и бесплатных приложений.
- Биллинг при продаже приложений.
- Запуск Google Play из любого приложения.
- Маркетинг и монетизация приложений.
- Другие магазины приложений Android.
- Другие популярные мобильные и интернет-платформы, на которые можно перенести приложения.
- Юмор Android-сообщества.

2.1. Введение

В следующих главах мы рассмотрим множество различных Android-приложений. После завершения разработки и тестирования собственных приложений (с помощью эмулятора или устройств Android) наступает следующий этап. На этом этапе осуществляется дистрибуция приложений (с помощью Google Play или других «ярмарок приложений»). В этой главе вашему вниманию предлагаются рекомендации по разработке интерфейса пользователя и лучшие методики по созданию выдающихся приложений. Рассматриваются способы регистрации на Google Play и создания учетной записи Google Checkout, обеспечивающие возможность продажи ваших приложений. Излагаются методики подготовки приложений к публикации и загрузка на Google Play. Обсуждается выбор

между распространением приложений на бесплатной основе или за определенную плату. Упоминаются ключевые ресурсы, применяемые для монетизации приложений. Здесь же вы найдете перечень ресурсов, которые могут использоваться для маркетинга приложений, и других популярных прикладных платформ, использующихся для переноса приложений Android. В главе также приведены ссылки на документацию для Android-разработчиков с дополнительной информацией.

2.2. Создание выдающихся Android-приложений

В связи с тем что на Google Play доступно более 200 000 приложений[1] возникает вопрос относительно условий, помогающих пользователям находить, загружать, использовать и рекомендовать другим пользователям созданное вами приложение. Что же делает приложение полезным, привлекательным и надежным? Запоминающееся имя, привлекательная пиктограмма и остроумное описание — все это способно привлечь внимание посетителей Google Play либо других «рынков приложений» Android именно к вашему приложению. Даже если пользователи загрузят ваше приложение, как сделать так, чтобы они его использовали регулярно и рекомендовали другим пользователям? Ответ на этот вопрос вы найдете в списке 2.1, где приведены характеристики, присущие выдающимся приложениям.

СПИСОК 2.1. ОСОБЕННОСТИ, ПРИСУЩИЕ ВЫДАЮЩИМСЯ ПРИЛОЖЕНИЯМ

Общие характеристики

- Совместимость с *будущими* версиями Android (developer.android.com/sdk/1.5_r3/upgrading.html#FutureProofYourApps).
- *Достаточная частота* обновлений, содержащих новые функции.
- *Корректная работа* и быстрое устранение ошибок.
- Соответствие стандартам, определенным в *соглашениях* по разработке графического интерфейса Android-приложений.
- *Способность к отклику и* нетребовательность к ресурсам (память, полоса пропускания Сети, заряд аккумуляторной батареи).
- *Новаторство* и *креативный подход*.
- *Устойчивость*, гарантирующая возможность регулярного использования приложения.
- Использование высококачественной графики, фотографий, анимации и видеороликов.
- *Интуитивность* и простота в применении (не требуется обширная пользовательская документация).
- *Обеспечение доступа* для людей с особыми потребностями (www.google.com/accessibility/).
- Возможности и средства *по обмену информацией о приложении* с другими пользователями (например, предоставление возможности отправлять сведения о достижениях в играх на Фейсбуке).

[1] googleblog.blogspot.com/2011/05/android-momentum-mobile-and-more-at.html.

- *Поддержка дополнительного контента* для приложений, управляемых контентом (дополнительные игровые уровни, загадки либо статьи).
- *Отсутствие требований избыточных прав доступа*.
- Встроенная возможность *широкого распространения*.

Характеристики выдающихся игр

- *Увлекательность*.
- *Вызов пользователям*.
- *Возрастающая сложность уровней*.
- Отображение набранных пользователем очков и *таблицы рекордов*.
- Поддержка *аудиовизуальной* обратной связи.
- Наличие режимов *однопользовательской*, *многопользовательской* и *сетевой* игры.
- Высококачественная *анимация.*
- Поддержка *схем управления* для различных устройств.

Свойства полезных утилит

- Наличие *полезной* функциональности и обеспечение точности информации.
- Обеспечение *более удобного* способа выполнения задач (например, поддержка списка запланированных задач, управление издержками).
- Обеспечение *лучшей информированности* пользователя.
- Предоставление информации о *текущих событиях* (курсы акций, последние новости, штормовые предупреждения, обзоры фильмов и пр.).
- Обеспечение доступа к *любимым веб-сайтам* пользователей (например, к сайтам интернет-магазинов, банков, социальных сетей и к другим сайтам).
- Улучшение персональной и коллективной *производительности*.

2.3. Лучшие методики для разработчиков Android-приложений

В разделе *Best Practices* документа *Android Developer's Guide* (сокращенное название *Dev Guide*) рассматриваются вопросы совместимости, поддержки нескольких экранов, указания по разработке интерфейса пользователя, а также разработки с учетом достижения производительности, способности к отклику и незаметности выполнения для пользователя. Также просмотрите общие указания по разработке приложений для мобильных устройств, доступные на других интернет-ресурсах (табл. 2.1).

Таблица 2.1. Интернет-ресурсы для разработчиков мобильных приложений

Название ресурса	URL-ссылка
Compatibility (Совместимость)	developer.android.com/guide/practices/compatibility.html
Supporting Multiple Screens (Поддержка нескольких экранов)	developer.android.com/guide/practices/screens_support.html

Название ресурса	URL-ссылка
User Interface Guidelines (Указания по разработке интерфейса пользователя)	developer.android.com/guide/practices/ui_guide lines/index.html
Designing for Performance (Проектирование с учетом производительности)	developer.android.com/guide/practices/design/performance.html
Designing for Responsiveness (Проектирование с учетом способности к отклику)	developer.android.com/guide/practices/design/responsiveness.html
Designing for Seamlessness (Проектирование с учетом незаметности выполнения приложения для пользователя)	developer.android.com/guide/practices/design/seamlessness.html

2.3.1. Совместимость

При разработке приложения Android следует идентифицировать устройства и версии операционной системы, на которых оно будет выполняться. Потребности приложения описаны с помощью элементов <uses-feature>, находящихся в файле манифеста (табл. 2.2.). Эти элементы позволяют Google Play осуществлять фильтрацию приложений так, чтобы их могли просматривать и загружать лишь пользователи, обладающие совместимыми устройствами.

Таблица 2.2. Дескрипторы свойств для спецификации аппаратных и программных требований, включаемые в файл манифеста (developer.android.com/guide/topics/manifest/uses-featureelement.html)

Свойство	Описатель
Оборудование	
Звук	android.hardware.audio.low_latency
Bluetooth	android.hardware.bluetooth
Камера	android.hardware.camera
Система автофокусировки камеры	android.hardware.camera.autofocus
Вспышка камеры	flash android.hardware.camera.flash
Фронтальная камера	android.hardware.camera.front
Определение местоположения с помощью Сети	android.hardware.location.network
GPS	android.hardware.location.gps
Микрофон	android.hardware.microphone
Коммуникации ближнего поля	android.hardware.nfc
Датчик акселерометра	android.hardware.sensor.accelerometer
Датчик барометра	android.hardware.sensor.barometer
Датчик компаса	android.hardware.sensor.compass

продолжение ➩

Таблица 2.2 *(продолжение)*

Свойство	Описатель
Датчик гироскопа	android.hardware.sensor.gyroscope
Датчик освещенности	android.hardware.sensor.light
Датчик приближения	android.hardware.sensor.proximity
Телефония	android.hardware.telephony
CDMA-телефония	android.hardware.telephony.cdma
GSM-телефония	android.hardware.telephony.gsm
Эмулированный сенсорный экран	android.hardware.faketouch
Сенсорный экран	android.hardware.touchscreen
Мультисенсорный экран (два или больше пальцев)	android.hardware.touchscreen.multitouch
Дискриминатор нескольких касаний (уникальная система отслеживания положений двух пальцев, используемых для выполнения вращательных жестов)	android.hardware.touchscreen.multitouch.distinct
Система распознавания касания экрана несколькими пальцами (распознает одновременное применение до пяти пальцев)	android.hardware.touchscreen.multitouch.jazzhand
Wi-Fi	android.hardware.wifi
Программное обеспечение	
Живые обои	android.software.live_wallpaper
SIP	android.software.sip
SIP/VoIP	android.software.sip.voip

Можно фильтровать продажи и загрузки разработанного вами приложения по странам и провайдерам мобильной связи. Например, ваше приложение может предназначаться только для подписчиков провайдера Verizon или для пользователей, проживающих в Великобритании. Эти «рыночные» фильтры могут устанавливаться после регистрации на Google Play, с помощью которого публикуются приложения. Приложения также могут динамически опрашивать устройства с целью определения его характеристик. Например, если приложение обладает функциями, предусматривающими использование камеры, но может обходиться и без камеры, оно может опрашивать устройство с целью определения доступности его камеры.

Чтобы получить информацию о *разработке приложений для нескольких устройств* и обеспечения работоспособности приложения после выхода *новых версий* операционной системы Android, обратитесь к веб-сайту developer.android.com/guide/practices/compatibility.html. Чтобы получить дополнительные сведения о фильтрах Market, позволяющих ограничить распространение приложения, обратитесь к веб-сайту developer.android.com/guide/appendix/market-filters.html.

2.3.2. Поддержка нескольких экранов

Библиотека Android SDK 1.6 или более старшая версия поддерживает несколько *размеров экранов* (измеряются по диагонали) *экранных разрешений* (количество пикселей вдоль каждой стороны экрана). Но вряд ли понадобится разрабатывать приложение, которое предназначено для экранов, имеющих любые размеры и разрешения (разве что в исследовательских целях).

На платформе Android поддерживаются четыре обобщенных значения размера (*маленький*, *обычный*, *большой* и *сверхбольшой*) и разрешения (*низкое*, *среднее*, *высокое* и *сверхвысокое*). В результате обеспечивается разработка приложений, предназначенных для использования на нескольких экранах. Эти значения размера и разрешения экрана можно использовать при разработке приложений, даже если фактические размеры экранов устройств Android отличаются. Можно также создать несколько ресурсов (например, макетов, пиктограмм и рисунков) с целью их масштабирования на соответствующих экранах. Когда пользователь выполняет приложение, Android автоматически отображает его на экране устройства, имеющего определенный размер и разрешение экрана, а также выбирает подходящие ресурсы (если указаны различные ресурсы для экранов с разными размерами). Можно также настроить значение элемента <supports-screens> из файла AndroidManifest.xml таким образом, чтобы указать различные размеры экрана, поддерживаемые приложением. Дополнительные сведения по этой теме можно найти в документе *Supporting Multiple Screens*, который доступен на веб-сайте developer.android.com/guide/practices/screens_support.html.

2.3.3. Советы по разработке интерфейса пользователя Android

В процессе разработки приложений Android рекомендуется придерживаться указаний по разработке пиктограмм, виджетов, деятельностей, задач и меню, изложенных в документе *Android User Interface Guidelines*, находящемся на веб-сайте developer.android.com/guide/practices/ui_guidelines/index.html.

Указания по разработке пиктограмм

В документе *Icon Design Guidelines* (Указания по разработке пиктограмм) приведены сведения, используемые при создании пиктограмм (таких как *пиктограмма запуска приложения*, *меню*, *панели состояния*, *вкладки*, *диалогового окна* и *списка*), а также спецификации дизайна для каждой пиктограммы (размеры, цвет, позиционирование, эффекты и другие спецификации). Также предлагается воспользоваться загружаемым пакетом Android Icon Templates Pack, который предлагает шаблоны для создания собственных пиктограмм приложений в Adobe Photoshop и Adobe Illustrator.

Указания по разработке виджетов

В документе *Widget Design Guidelines* (Указания по разработке виджетов) приведены спецификации, применяемые в процессе создания *виджетов* для отображения на пользовательском главном экране динамически изменяющейся информации, например текущей погоды, курсов акций, цен на бензин и последних новостей (табл. 2.3). Виджеты могут быть автономными (см. главу 14, где разрабатывается виджет Weather Viewer App), но чаще всего они выступают в качестве дополнительной функции приложения,

повышающей его ценность в глазах пользователя. Например, приложение ScoreCenter, предлагаемое компанией ESPN, включает виджет, предназначенный для отслеживания на главном экране пользователя результатов игр его любимых спортивных команд. Это гораздо удобнее, чем запускать приложение каждый раз, когда нужно просмотреть результаты последней игры. Пользователь может самостоятельно выбирать отображение виджета приложения на своем главном экране.

Таблица 2.3. Популярные виджеты Android

Виджет	Набор функций
ESPN® ScoreCenter	Отслеживание результатов игр любимых спортивных команд
Pandora Radio	Управление настроенными интернет-радиостанциями Pandora (например, приостанавливать воспроизведение или пропускать)
WeatherBug Elite	Трехдневный прогноз погоды и виджет карты погоды
Twidroyd PRO	Следуйте за вашими любимыми твитами
Shazam Encore	Простое тегирование, рассылка и приобретение музыки
Weather & Toggle Widget	Виджеты времени, погоды и настроек. С помощью последних виджетов можно изменять настройки смартфона Android (например, яркость, включение/отключение Wi-Fi и ряд других)
BatteryLife	Настраиваемый виджет, обеспечивающий контроль заряда аккумуляторной батареи устройства
System Info	Мониторинг системной информации, например заряда аккумуляторной батареи, объема свободной памяти (оперативной, внутренней и на карте памяти SD) и ряд других параметров системы
Stock Alert	Отслеживание цен акций, курсов валют, цен на основные товары и фьючерсы
The Coupons App	Купоны на скидки для местных ресторанов, магазинов и автозаправочных станций, отображаемые в режиме реального времени
Favorite Quotes	Виджеты, отслеживающие ежедневные и случайные квоты
ecoTips	Экологические советы с веб-сайта Wildlife Fund
Difficult Logic Riddles Pro	Математические и логические задачи (приведены подсказки и ответы)
App Protector Pro	Блокировка любого приложения, выполняемого на телефоне (например, SMS, Market и др.)
Android Agenda Widget	Отображение событий для календаря, установленного на устройстве, Google Calendar и других календарей

Указания по проектированию деятельностей и задач

В документе *Activity and Task Design Guidelines* (Указания по проектированию деятельностей и задач) рассматриваются следующие вопросы:

- **Деятельности.** Повторно используемые компоненты, применяемые для создания интерфейса пользователя. Эти компоненты предназначены для выполнения определенных задач, таких как поиск, просмотр информации и набор телефонного номера. Каждая отдельная деятельность часто связана со своим экраном приложения. Деятельности будут рассмотрены в главе 4.
- **Стек деятельностей.** Обратная хронологическая история всех деятельностей, позволяющая пользователю перейти к предыдущей деятельности с помощью кнопки Back (Назад).
- **Задачи.** Набор деятельностей, с помощью которых пользователи могут достигать определенных целей в рамках одного или нескольких приложений.

Указания по проектированию меню

В документе *Menu Design Guidelines* (Указания по разработке меню) рассматривается разработка меню Options (Параметры) и Context (Контекст). Доступ к меню Options можно получить с помощью кнопки Menu (Меню) устройства. Эта кнопка открывает доступ к операциям и действиям, выполняемым на текущем экране приложения. Например, в результате выбора меню Options для приложения Messaging отображается меню, включающее пиктограммы Compose (Создать), Delete Threads (Удалить беседы), Search (Поиск) и Settings (Настройки). В результате выбора меню Context из сообщения в приложении Messaging (выбирается путем касания и удержания пальца на сообщении, отображающемся на сенсорном экране, — длинное касание) появляется меню параметров, специфичных для выбранного сообщения, включая такие параметры, как Select all (Выделить все), Select text (Выделить текст), Cut all (Вырезать все), Copy all (Скопировать все), Paste (Вставить) и Input method (Метод ввода).

В списках 2.2 и 2.3 приведены советы по созданию интерфейсов пользователя, в том числе советы, помогающие разработать приложения, которые отзываются на все действия со стороны пользователя, а также эффективно и беспрепятственно выполняются на мобильных устройствах. Примеры лучших методик кодирования будут рассмотрены в других главах книги.

СПИСОК 2.2. СОВЕТЫ ПО РАЗРАБОТКЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Общие советы:

- Прочтите руководство *Best Practices* от *Dev Guide* (в том числе и *User Interface Guidelines*).
- Учитывайте *причины*, побуждающие пользователя применять ваше приложение.
- При разработке приложения принимайте во внимание *цели* его создания.
- Смоделируйте приложение, используя *реальные условия* его применения.
- Установите *обратную связь* с действиями пользователя (например, воспользуйтесь индикаторами, такими как *индикатор выполнения*, которые позволяют увидеть ход выполнения приложения).

- Поддерживайте стандартные *жесты* Android (см. табл. 1.5).
- *Читайте отзывы пользователей*, содержащие информацию о допущенных ошибках, и вносите соответствующие исправления.
- Поддерживайте взаимодействие между приложениями (см. developer.motorola.com/docstools/library/Best_Practices_for_User_Interfaces/).

Проектирование интерфейса пользователя:

- Создавайте *интуитивно* понятные приложения — пользователь должен получить желаемое с минимальными усилиями.
- Позаботьтесь об *эстетической привлекательности* — используйте привлекательные цвета, высококачественную графику и другие подобные элементы оформления.
- *Не загромождайте экран*.
- Старайтесь использовать *списки*, элементы которых можно выбрать касанием (или выделением), вместо ввода информации.
- По возможности применяйте *стандартные кнопки и пиктограммы*, поддерживаемые в Android.
- Если применяются *пользовательские пиктограммы*, сделайте так, чтобы они *отличались* от системных пиктограмм Android.
- Элементы интерфейса пользователя должны соответствовать размеру пальца.
- Все размеры шрифтов следует определять с помощью пикселей, независимых от разрешения (SP); используйте пиксели, независимые от разрешения (DIP или DP), и для указания размеров других элементов (см. stackoverflow.com/questions/2025282/difference-of-pxdp-dip-and-sp-in-android).
- Поддерживайте изменение ориентации экрана между *портретной* (устройство сориентировано верхней панелью вверх) и *альбомной*, когда устройство «лежит на боку» либо подключена физическая клавиатура.
- Предусматривайте возможность выполнения приложения на *экранах, имеющих различные размеры* (см. developer.android.com/guide/practices/screens_support.html), и разных устройствах.

СПИСОК 2.3. РАЗРАБОТКА ПРОИЗВОДИТЕЛЬНЫХ, СПОСОБНЫХ К ОТКЛИКУ И ВЫПОЛНЯЮЩИХСЯ НЕЗАМЕТНО ДЛЯ ПОЛЬЗОВАТЕЛЯ ПРИЛОЖЕНИЙ

Производительность (developer.android.com/guide/practices/design/performance.html):

- Приложения должны быть *эффективными*, поскольку устройства Android имеют аккумуляторные батареи невысокой емкости, ограниченную вычислительную мощность и небольшую память.
- *Избегайте длительных задач (таких как загрузка больших файлов или операции с базой данных) в потоке UI*, чтобы не «тормозить» приложение.
- *Удаляйте ненужные файлы* из кэш-памяти.
- Предусмотрите способ обработки приложением при *разорванном или недоступном сетевом подключении* (например, отобразите соответствующее сообщение для пользователя).

- Приложение должно извещать пользователя о любых действия, которые могут привести к *дополнительным платежам провайдеру мобильной связи* (например, использование дополнительных служб передачи данных, рассылка сообщений SMS или MMS).
- Многие устройства имеют ограниченный объем памяти, предназначенной для хранения приложений и данных. Если приложение не использует защищенные данные, предусмотрите возможность их хранения на карте памяти SD (при ее наличии).

Способность к отклику (developer.android.com/guide/practices/design/responsiveness.html):

- Создавайте *эффективный* код, благодаря которому приложения будут *быстрыми* и *способными к отклику*.
- Если требуется время для загрузки приложения, используйте *всплывающий экран*, — изображение, которое отображается после того, как пользователь нажмет пиктограмму. Благодаря этому экрану пользователь получает немедленный отклик во время загрузки приложения. Всплывающий экран обычно создается на основе элементов интерфейса пользователя (зачастую элементы интерфейса, отображающиеся на фоне какой-нибудь картинки). Во время загрузки приложения можно также отображать индикатор выполнения.

Незаметность выполнения приложения для пользователя (developer.android.com/guide/practices/design/seamlessness.html):

- При разработке приложения учитывайте необходимость корректной обработки изменений конфигурации, например изменения ориентации и выдвижения/скрытия физической клавиатуры.
- *Сохраняйте пользовательские данные* перед переключением приложения в фоновый режим.
- Используйте возможности ContentProvider для *обмена данными* между приложениями, которые установлены и выполняются на устройстве.
- Для отправки уведомлений пользователю используйте NotificationManager.
- Не запускайте Activity UI из фонового режима.
- Проектируйте приложения с учетом их выполнения на нескольких устройствах — предусмотрите возможность ввода данных с помощью сенсорного экрана и клавиатуры, а также использование экранов, имеющих различные размеры и разрешения.

Учитывайте требования по обеспечению доступности

В состав платформы Android включены инструментальные средства, предназначенные для разработки приложений, ориентированных на пользователей с особыми потребностями (например, слабовидящих). Функция преобразования текста в речь (Text-to-Speech, TTS), которая поддерживает английский, испанский, французский, немецкий и итальянский языки, позволяет «проговаривать» текстовые строки. Можно также включить обратную связь путем воспроизведения звука (для слабовидящих пользователей) и вибрации (для слабослышащих пользователей).

Локализация

Если вы хотите сделать доступным ваше приложение для пользователей из различных стран, не забудьте о необходимости локализации. Например, если вы намереваетесь распространять приложение во Франции, переведите относящиеся к нему ресурсы (текст и звуковые файлы) на французский язык. Возможно, вам придется выбрать различные цвета, графику и звук, основываясь на *локали*. Для каждой локали следует создать отдельный настроенный набор ресурсов приложения. Как только пользователь запускает приложение, Android автоматически находит и загружает ресурсы, соответствующие локали для устройства. Дополнительные сведения о настройке различных каталогов ресурсов, применяемых для локализации приложений, можно найти на веб-сайте developer.android.com/guide/topics/resources/localization.html.

2.4. Регистрация на Google Play

Чтобы иметь возможность публиковать приложения на Google Play, следует создать учетную запись на веб-странице play.google.com/apps/publish/signup.

В процессе регистрации придется один раз заплатить регистрационный взнос. В отличие от других мобильных платформ, Google Play не требует выполнения *процедуры одобрения в процессе загрузки приложений*. Обратите внимание на необходимость выполнения требований, изложенных в документе *Android Market Content Policy for Developers*. Если приложение нарушает принципы этой политики, оно может быть удалено. В случае повторяющихся или серьезных нарушений положений этого документа учетная запись может быть принудительно удалена (список 2.4).

СПИСОК 2.4. ПЕРЕЧЕНЬ ВОЗМОЖНЫХ НАРУШЕНИЙ ПОЛОЖЕНИЙ ДОКУМЕНТА ANDROID MARKET CONTENT POLICY FOR DEVELOPERS

- Нарушение прав владельцев интеллектуальной собственности (торговые марки, патенты и копирайты).
- Пропаганда ненависти и насилия.
- Поддержка контента порнографического или непристойного содержания либо любого другого контента, недопустимого для просмотра лицами до 18 лет.
- Нарушение условий обслуживания, предлагаемых провайдером.
- Нелегальный контент.
- Нарушение неприкосновенности частной жизни пользователя.
- Посягательство на область действия услуг других предпринимателей.
- Повреждение личных данных или устройства пользователя.
- Намеренные действия, направленные на рост расходов пользователя мобильной сети передачи данных или провайдера беспроводной сети.
- Намеренный ввод пользователя в заблуждение относительно назначения приложения.
- Регистрация под чужим именем.
- Организация азартных игр.

2.5. Создание учетной записи Google Checkout Merchant

Чтобы иметь возможность продавать приложения на Google Play, следует создать учетную запись Google Checkout Merchant. Возможность создания этой учетной записи доступна для продавцов приложений на Google Play, находящихся в 29 странах на момент выхода этой книги из печати (табл. 2.4)[1]. После регистрации и входа на Google Play по адресу market.android.com/publish/ щелкните на ссылке Setup Merchant Account (Настройка учетной записи торговца). Затем выполните следующие действия:

- укажите контактную информацию, предназначенную для связи Google с вами;
- укажите контактную информацию, предназначенную для пользователей вашего приложения;
- введите финансовую информацию, которую будет использовать Google для выполнения операций с вашим счетом;
- согласитесь с условиями обслуживания (Terms of Service), которые описывают особенности обслуживания, допустимые транзакции, запрещенные действия, плату за обслуживание, сроки платежей и другие условия.

Таблица 2.4. Страны, в которых поддерживаются учетные записи Google Checkout Merchant

Аргентина	Дания	Израиль	Норвегия	Швеция
Австрия	Финляндия	Италия	Португалия	Швейцария
Австралия	Франция	Япония	Россия	Тайвань
Бельгия	Германия	Мексика	Сингапур	Великобритания
Бразилия	Гонконг	Нидерланды	Испания	США
Канада	Ирландия	Новая Зеландия	Южная Корея	

С помощью Google Checkout выполняется обработка платежей и обеспечивается защита от мошеннических действий. В случае если покупатель не заплатил за приложение, вы можете получить компенсацию[2], но при этом придется заплатить 30 % от суммы транзакции Google Play. После настройки учетной записи Google Checkout ее возможности не ограничиваются продажами приложений. Подобно учетной записи PayPal, учетная запись Google Checkout может применяться в качестве платежного сервиса при выполнении транзакций в Интернете. В будущем планируется появление на Google Play других платежных сервисов, например PayPal.

[1] checkout.google.com/support/sell/bin/answer.py?answer=150324&cbid=-eqo3objy740w&src=cb&lev=%20index.

[2] checkout.google.com/termsOfService?type=SELLER.

2.6. Файл AndroidManifest.xml

Файл AndroidManifest.xml, который также называется *манифестом*, включает информацию, которая применяется для выполнения приложения на платформе Android и корректной фильтрации на Google Play. Благодаря этому можно скрывать приложение от пользователей, просматривающих Google Play и использующих устройства, несовместимые с этим приложением. Например, пользователи устройств без встроенной камеры не смогут получить доступ к приложениям, требующим камеру (это требование указывается в манифесте приложения). Манифест автоматически создается модулем ADT Plugin для Eclipse, и перед загрузкой приложения на Google Play потребуется вручную добавить в манифест необходимую информацию. Манифест, представляющий собой XML-файл, можно редактировать вручную либо с помощью программы Android Manifest Editor. Эта программа включена в состав модуля ADT Plugin для Eclipse.

Чтобы получить доступ к редактору Android Manifest Editor в среде Eclipse, выберите вкладку Packages Explorer (Просмотр пакетов) и дважды щелкните на файле AndroidManifest.xml, находящемся в папке приложения. После этого файл откроется в рабочей среде Eclipse. Выберите вкладку Manifest (Манифест), находящуюся в нижней части страницы рабочей среды, для перехода к странице Manifest General Attributes (Общие атрибуты манифеста). На этой странице следует указать основные сведения о приложении, такие как названия пакетов, номера версий и элементы. В табл. 2.5 перечислены некоторые общие элементы, включенные в манифест. Полный перечень элементов можно найти на веб-сайте developer.android.com/guide/topics/manifest/manifest-intro.htm.l

Как только необходимые сведения будут внесены в манифест, вернитесь на страницу Manifest General Attributes для подготовки приложения к распространению (см. раздел 2.8).

Таблица 2.5. Некоторые общие элементы манифеста приложения

Элемент	Описание
Uses Feature (Применения свойств)	Определяет свойства, используемые приложением. См. раздел 2.3.1, «Совместимость»
Protected Broadcast (Защищенная трансляция)	Определяет название защищенной трансляции, которая позволяет приложению объявлять, что только оно может отсылать транслированное содержимое
Supports Screens (Поддержка экранов)	Определяет физические размеры экрана (Small (маленький), Normal (обычный), Large (большой), XLarge (сверхбольшой), Resizeable (с изменяемыми размерами)) и разрешения экрана (плотность пикселей на экране), поддерживаемые приложением. Для каждого параметра выберите значение true или false
Uses Configuration (Применения конфигурации)	Определяет требования к аппаратным средствам со стороны приложения. Этот элемент может принимать следующие значения: Touch screen (Сенсорный экран), Keyboard type (Тип клавиатуры), Hard keyboard (Физическая клавиатура), Navigation (Навигация) (например, трекбол или колесико) или Five way nav key (Пятипозиционная

Элемент	Описание
	навигационная клавиша) (трекбол или клавиша, с помощью которой можно перемещаться верх, вниз, вправо или влево, а также выбирать элемент на экране)
Uses SDK (Применения SDK)	Функции SDK требуются для корректного выполнения приложения (функции, специфичные для платформ Android 2.3, 3.0 и более старших). Обратите внимание, что в результате установки этого флага приложение, разработанное на основе текущей библиотеки Android SDK, может выполняться на устройстве, на котором установлена более ранняя версия SDK. При этом не производится вызов API, не поддерживаемых в этой ранней версии

На вкладке Application (Приложение), находящейся в нижней части окна приложения, определяются специфичные для приложения атрибуты, в том числе пиктограмма, атрибуты описания, разрешения, отладки и ряд других. На вкладке Permissions (Разрешения) определяется, должно ли приложение использовать защищенные свойства устройства (свойства, для доступа к которым требуются разрешения, например написание SMS-сообщений, настройка обоев или доступа к местоположению). Перед установкой приложения Google Play отображает перечень разрешений, требуемых приложением. Следует запрашивать только те разрешения, которые приложение может обрабатывать корректно. Список разрешений приводится на веб-сайте developer.android.com/reference/android/Manifest.permission.html. Подробное рассмотрение редактирования файла манифеста ожидает вас в разделе 2.7.

2.7. Подготовка приложений к публикации

В разделе *Preparing to Publish: A Checklist* (Подготовка к публикации: контрольный список) документа *Dev Guide*, который находится на веб-сайте developer.android.com/guide/publishing/preparing.html, перечислены рекомендации для подготовки приложения для публикации на Google Play, например:

- протестируйте приложение на устройствах Android;
- включите в комплект поставки приложения (необязательно) лицензионное соглашение с конечным пользователем (End User License Agreement);
- добавьте пиктограммы и надписи в манифест приложения;
- отключите регистрацию и отладку;
- создайте несколько версий приложения (например, 1.0, 1.1, 2.0, 2.3, 3.0);
- получите криптографический ключ, используемый для цифровой подписи приложения;
- скомпилируйте приложение;
- подпишите приложение.

Некоторые из элементов этого списка будут рассмотрены в следующих разделах.

Тестирование приложения

Прежде чем публиковать приложение на Google Play, выполните его всестороннее тестирование на различных устройствах, чтобы убедиться в его корректной работе. Необходимость в этом возникает в связи с тем, что приложение, которое может корректно выполняться на компьютере с помощью эмулятора, будет сбоить при выполнении на том или ином устройстве Android. В списке 2.5 перечислены функциональнее свойства Android, которые недоступны для эмулятора.

СПИСОК 2.5. ФУНКЦИИ ANDROID, КОТОРЫЕ НЕ ПОДДЕРЖИВАЮТСЯ ЭМУЛЯТОРОМ

- Выполнение и получение реальных телефонных звонков (эмулятор может лишь имитировать звонки).
- USB-подключения.
- Камера и видеосъемка.
- Наушники, подключаемые к устройству.
- Идентификация подключенного состояния телефона.
- Идентификация заряда батареи и состояние, в котором заряжается батарея.
- Определение вставки/извлечения SD-карты.
- Bluetooth.
- Коммуникации ближнего поля.
- Датчики (акселерометр, барометр, компас, датчик освещенности, датчик приближения).
- OpenGL ES 2.0 (and non-software rendered OpenGL).

Чтобы перевести устройство Android в режим тестирования и отладки приложений, выполните команды Settings Applications ▸ Development (Настройки приложения ▸ Разработка) и установите флажок USB (Universal Serial Bus) Debugging (Отладка USB).

Соглашение с конечным пользователем

Можно включить в приложение соглашение с конечным пользователем (End User License Agreement, EULA). С помощью этого соглашения определяется использование приложения конечным пользователем. В этом документе обычно определяются условия использования, ограничения по повторному распределению и реверсивному инжинирингу, ответственность разработчика приложения, соответствие правовым нормам и ряд других условий. Посоветуйтесь с юристом по поводу соглашения EULA для вашего приложения. Чтобы просмотреть образец EULA, обратитесь к веб-сайту www.developer-resource.com/sample-eula.htm.

Пиктограммы и надписи

Создайте пиктограммы для приложения и текстовые надписи (названия), которые будут отображаться в окне Google Play и на экране устройства. В качестве пиктограммы может использоваться логотип компании, экранный снимок приложения или

пользовательское изображение. Для экранов с различным разрешением создавайте пиктограммы, имеющие следующие размеры:

- экраны с высоким разрешением: 72×72 пикселя;
- экраны со средним разрешением: 48×48 пикселей;
- экраны с низким разрешением: 36×36 пикселей.

Для отображения в окне Google Play применяются пиктограммы высокого разрешения[1]. Эти пиктограммы должны соответствовать следующим условиям:

- 512×512 пикселей;
- 32-битовый PNG с прозрачностью;
- максимальный размер 1024 Кбайт.

Чтобы ознакомиться с дополнительными спецификациями и лучшими методиками, прочтите документ *Icon Design Guidelines* (Указания по созданию пиктограмм), который находится на веб-сайте developer.android.com/guide/practices/ui_guidelines/icon_design.html. Обратитесь к услугам опытного графического дизайнера, который поможет вам разработать профессиональные и красивые пиктограммы (табл. 2.6). Стоимость услуг по разработке пиктограммы для приложений в США варьируется от 65 до 400 долларов (или даже больше). После завершения создания пиктограммы и надписи укажите их в манифесте приложения. Для выполнения этой задачи откройте окно Android Manifest Editor и выберите вкладку Application, находящуюся в нижней части окна.

Таблица 2.6. Компании, предоставляющие услуги по созданию пиктограмм

Компания	URL-ссылка	Предоставляемые услуги
glyFX	www.glyfx.com/index.html	Проектирование пользовательских пиктограмм и предоставление возможности загрузки готовых бесплатных пиктограмм
Androidicons	www.androidicons.com/	Проектирование пользовательских пиктограмм и предоставление возможности загрузки множества готовых бесплатных пиктограмм
Iconiza	www.iconiza.com/portfolio/appicon.html	Разработка пользовательских пиктограмм по фиксированной цене
Aha-Soft	www.aha-soft.com/icon-design.htm	Разработка пользовательских пиктограмм по фиксированной цене
Elance®	www.elance.com	Привлечение дизайнеров пиктограмм, работающих на условиях фриланса

[1] market.android.com/support/bin/answer.py?answer=1078870.

Отключение регистрации и отладки

Перед публикацией приложения отключите режим отладки. Выберите вкладку Application в окне Android Manifest Editor и присвойте атрибуту Debuggable значение false. Удалите лишние файлы (такие как файлы журналов и файлы резервных копий).

Управление версиями приложения

Включите в приложение номер версии (отображаемый для пользователей) и код версии (целочисленное значение, используемое на Google Play), а также продумайте стратегию нумерации обновлений. Например, код первой версии приложения может быть 1.0, незначительные обновления могут иметь номера 1.1 и 1.2, а ближайшее большое обновление — 2.0. Дополнительные сведения о контроле версий можно найти в документе *Versioning Your Applications* (Управление версиями приложения), опубликованном на веб-сайте developer.android.com/guide/publishing/versioning.html.

Сжатие, оптимизация и скрытие кода приложения

Служба лицензирования Google Play позволяет создавать политики лицензирования, контролирующие доступ к платным приложениям. Например, можно воспользоваться политикой лицензирования для контроля частоты проверки приложения на сервер, количество устройств, на которые одновременно разрешается установка приложения, а также для программирования действий, происходящих в случае идентификации нелицензионного приложения. Чтобы получить дополнительные сведения о службе лицензирования, посетите веб-сайт developer.android.com/guide/publishing/licensing.html.

Помимо создания политики лицензирования следует зашифровать приложения, загруженные на Google Play, чтобы сделать невозможным обратный инжиниринг кода и обеспечить дополнительную защиту приложений. С помощью инструмента ProGuard, используемого при создании приложений в режиме выпуска, уменьшается размер файла .apk, а также выполняется оптимизация и скрытие кода. Чтобы получить дополнительные сведения о настройке и применении инструмента ProGuard, обратитесь к веб-сайту developer.android.com/guide/developing/tools/proguard.html.

Для получения дополнительных сведений о защите приложений от пиратского использования кода, скрытия кода и других методиках, направленных на повышение степени защиты приложений, обратитесь к веб-сайту android-developers.blogspot.com/2010/09/securing-android-lvlapplications.html.

Получение закрытого ключа, применяемого для цифровой подписи приложения

Прежде чем выгружать приложение на устройство, на Google Play либо на другой «рынок приложений», нужно *подписать цифровой подписью* файл .apk (файл пакета приложения Android). При этом используется *цифровой сертификат*, который идентифицирует вас как автора приложения. Цифровой сертификат содержит имя пользователя или компании, контактную информацию и другие сведения. Можно воспользоваться технологией автоматического подписывания с помощью частного ключа (то есть защищенного пароля, используемого для шифрования сертификата). При этом не нужно приобретать сертификат у независимого центра сертификации (хотя

и возможно). В процессе разработки приложения Eclipse автоматически добавляет цифровую подпись в приложение, что позволяет выполнять созданное приложение на тестовых устройствах. Этот цифровой сертификат не может применяться на Google Play. Библиотека Java Development Kit (JDK) включает инструменты, используемые для подписывания приложений. Инструмент Keytool генерирует закрытый ключ, а Jarsigner используется для подписывания .apk файлов. При выполнении приложений в среде Eclipse инструменты создания выполняемых модулей, включенные в состав модуля ADT Plugin, автоматически обращаются к инструменту Keytool для подписывания файла .apk (при этом не придется вводить пароль). Затем используется инструмент zipalign, который оптимизирует использование памяти приложением.

Если применяется среда Eclipse вместе с подключаемым модулем ADT Plugin, можно воспользоваться модулем Export Wizard для компиляции приложения, генерирования закрытого ключа и подписывания файла .apk в режиме выпуска.

1. Выберите проект в окне Package Explorer и выполните команды File ▸ Export (Файл ▸ Экспорт).
2. Дважды щелкните на папке Android, чтобы открыть ее, затем выберите параметр Export Android Application (Экспорт приложения Android) и щелкните на кнопке Next (Далее).
3. Выберите проект (например, ваше приложение) для экспорта и щелкните на кнопке Next.
4. Установите переключатель Create new keystore (Создать новое хранилище ключей). Перейдите в папку Location (Местоположение), где хранятся цифровой сертификат и закрытый ключ (например, c:\android\keystore). Создайте защищенный пароль (Password), Подтвердите (Confirm) пароль, затем щелкните на кнопке Next, чтобы перейти к графическому интерфейсу Key Creation (Создание ключа).
5. В поле Alias (Псевдоним) введите уникальное имя ключа (например, «releasekey»). Обратите внимание, что используются только первые восемь символов псевдонима. В поле Password введите защищенный пароль для ключа, затем повторно введите его в поле Confirm (Подтверждение). В поле Validity (Действительность) укажите количество лет, в течение которых будет действителен ключ. Согласно требованиям Google Play, закрытый ключ будет действительным до 22 октября 2033 года. Таким образом Google предполагает, что ключ будет действительным более 25 лет (дольше, чем время существования типичного приложения). В результате все обновленные версии приложения подписываются одним и тем же ключом. *Если подписать обновленную версию другим ключом, у пользователей могут возникнуть проблемы с обновлением приложения*. В следующих полях введите персональную информацию, в том числе имя (First Name) и фамилию (Last Name), название отдела (Organizational Unit), компании (Organization), города (City) либо местоположения (Locality), штата (State) или провинции (Province) и двухбуквенный код страны (Country Code), например US. Щелкните на кнопке Next.

Дополнительные сведения о подписывании приложений можно найти на веб-сайте developer.android.com/guide/publishing/app-signing.html.

Экранные снимки

Сделайте как минимум два экранных снимка приложения для включения в описание приложения, опубликованное на Google Play (табл. 2.7). Эти снимки дадут представление о самом приложении еще до его загрузки. Выберите самые привлекательные снимки, отражающие возможности приложения. При создании снимков используйте эмулятор, у которого в строке состояния отсутствуют дополнительные пиктограммы и не применяются пользовательские экранные оболочки, которые могут ввести в заблуждение пользователей либо просто вызвать у них чувство раздражения. После загрузки приложения на Google Play можно добавить URL-ссылку презентационного видеоролика.

Таблица 2.7. Спецификации экранных снимков

Спецификация	**Описание**
Размер	320×480 (ширина и высота) пикселей или 480×854 (ширина и высота) пикселей (изображения с альбомной ориентацией следует соответствующим образом обрезать)
Формат	24-битовый формат PNG или JPEG без эффектов прозрачности
Изображение	Занимающее максимальную площадь экрана без четко выраженных границ

Для отладки приложений, выполняющихся на реальных устройствах, используется служба Dalvik Debug Monitor Service (DDMS), которая устанавливается одновременно с подключаемым модулем ADT Plugin для Eclipse. С помощью DDMS можно делать экранные снимки для приложения, выполняемого на реальном устройстве. Выполните следующие действия:

1. Запустите приложение на устройстве (см. последнюю часть раздела 1.11).
2. В среде Eclipse выполните команды Window ▸ Open Perspective ▸ DDMS (Окно ▸ Открыть представление ▸ DDMS). В результате откроется доступ к инструментам DDMS.
3. В окне Devices (рис. 2.1) выберите устройство, для которого будут делаться экранные снимки.

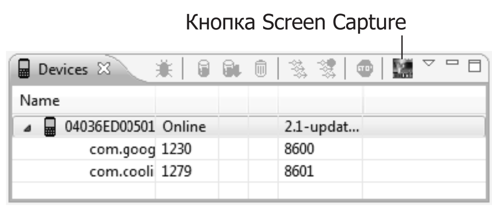


Рис. 2.1. Окно Devices в представлении DDMS

4. Щелкните на кнопке Screen Capture для отображения окна Device Screen Capture (Экранный снимок устройства), показанного на рис. 2.2.

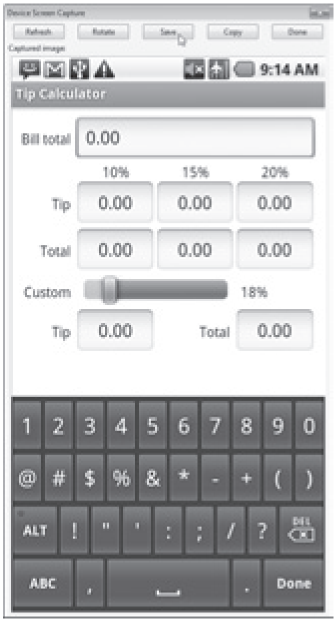


Рис. 2.2. В окне Device Screen Capture отображается экранный снимок приложения Tip Calculator (глава 4)

5. В случае удачного экранного снимка щелкните на кнопке Save (Сохранить) для сохранения полученного изображения.

Если нужно внести изменения в экранный снимок перед сохранением изображения, измените соответствующим образом параметры устройства, затем нажмите кнопку Refresh (Обновить), которая отображается в окне Device Screen Capture, для выполнения повторного снимка экрана.

2.8. Загрузка приложений на Google Play

Как только будут подготовлены все файлы и вы будете готовы к загрузке приложения, выполните шаги, описанные в документе developer.android.com/guide/publishing/publishing.html.

Зарегистрируйтесь на Google Play, market.android.com/publish (см. раздел 2.4), и щелкните на кнопке Upload Application (Загрузить приложение), чтобы начать процесс загрузки. В следующих разделах этот процесс описывается подробно.

Обновление ресурсов

1. App .apk file (файл .apk приложения). Щелкните на кнопке Choose File (Выбрать файл), чтобы выбрать файл пакета приложения Android (.apk) file, который включает файлы кода приложения (файлы .dex), ресурсы, файлы ресурсов приложения и файл манифеста. Щелкните на кнопке Upload (Загрузить).

2. Screenshots (Экранные снимки). Щелкните на кнопке Choose File и выберите как минимум два экранных снимка приложения, которые будут опубликованы на Google Play. После выбора экранных снимков щелкните на кнопке Upload.
3. High-resolution app icon (Пиктограмма приложения высокого разрешения). Щелкните на кнопке Choose File и выберите пиктограмму приложения с разрешением 512×512 пикселей, которая будет опубликована на Google Play затем щелкните на кнопке Upload.
4. Promotional graphic (Промо-графика) (необязательно). Можно загрузить промо-графику на Google Play, которая будет использована Google для продвижения вашего приложения (например, в разделе популярных приложений Google Play). Эта графика должна иметь разрешение 180×120 (ширина × высота) пикселей, 24-битовый формат PNG или JPEG и не использовать эффекты прозрачности. Графика должна занимать всю выделенную для нее область экрана и не иметь ярко выраженных границ. Для выбора изображения щелкните на кнопке Choose File, а затем щелкните на кнопке Upload.
5. Feature Graphic (Популярная графика) (необязательно). Графика, отображаемая в разделе Featured (Популярные приложения) на Google Play. Эта графика должна иметь разрешение 1024×500 (ширина × высота) пикселей, 24-битовый формат PNG или JPEG без эффектов прозрачности[1]. Выберите изображение, щелкнув на кнопке Choose File, затем щелкните на кнопке Upload.
6. Promotional video (Промо-видео) (необязательно). Можно включить URL-ссылку на промо-видеоролик для вашего приложения (например, ссылку на видеоролик YouTube, демонстрирующий возможности приложения).
7. Marketing opt-out (Запретить маркетинг). Установите флажок Marketing opt-out (Запретить маркетинг), если не хотите, чтобы Google рекламировал ваше приложение за пределами Google Play либо других сайтов Google.

Сведения о приложении

1. Language (Язык). По умолчанию сведения о приложении отображаются на английском языке. Если нужно отображать сведения о приложении на дополнительных языках, щелкните на ссылке add language (добавить язык) и установите флажки для дополнительных языков (табл. 2.8), а затем щелкните на кнопке OK. Каждый выбранный язык отображается в виде гиперссылки в разделе Language (Язык) в окне Listing Details (Сведения о приложении). Затем щелкайте на каждом языке, чтобы добавить переведенный заголовок, описание и промо-текст.

Таблица 2.8. Языки, используемые для описания приложения на Google Play

Французский	Испанский	Чешский	Японский	Шведский	Хинди
Немецкий	Голландский	Португальский	Корейский	Норвежский	Иврит
Итальянский	Польский	Тайваньский	Русский	Датский	Финский

[1] market.android.com/support/bin/answer.py?hl=en&answer=1078870.

2. Title (Заголовок). Заголовок приложения, отображаемый на Google Play (максимум 30 символов). Этот заголовок должен быть уникальным для каждого приложения Android.
3. Description (Описание). Описание приложения и его свойств (максимум 4000 символов). В последнем разделе описания рекомендуется обосновать обязательность каждого разрешения и продемонстрировать, каким образом оно используется.
4. Recent changes (Последние изменения). Обзор изменений в последней версии приложения (максимум 500 символов).
5. Promo text (Промо-текст). Рекламный текст, используемый для маркетинга приложения (максимум 80 символов).
6. App type (Тип приложения). Выберите тип приложения Applications (приложения) или Games (Игры).
7. Category (Категория). Выберите категорию (см. табл. 1.15), которая соответствует вашей игре или другому приложению.
8. Price (Цена). По умолчанию выбирается значение Free (Бесплатно). Если вы планируете продавать приложение, щелкните ссылку Setup a Merchant Account at Google Checkout (Установить торговую учетную запись на Google Checkout).

Параметры публикации

1. Content rating (Рейтинг контента). Выберите один из следующих параметров: Mature (Взрослые), Teen (Подростки), Pre-teen (Дети старшего возраста) или All (Все). Дополнительные сведения по этой теме можно найти в документах Google Play *Developer Program Policies* и *Content Rating Guidelines* на веб-сайте market.android.com/support/bin/answer.py?answer=188189.
2. Locations (Местоположения). По умолчанию выбирается параметр All Locations (Все местоположения). Это означает, что приложение будет отображаться во всех местоположениях Google Play (в настоящее время и в будущем). Чтобы выбрать отдельные местоположения Google Play, в которых будет отображаться ваше приложение, отмените установку флажка All Locations, в результате чего отобразится список стран. Выберите требуемые страны.

Контактная информация

1. Website (Веб-сайт). На Google Play может отображаться URL-ссылка на ваш веб-сайт. По возможности включите прямую ссылку на страницу приложения, чтобы пользователи, которые хотят загрузить ваше приложение, могли найти дополнительные сведения: рекламную информацию, перечни свойств и функций, дополнительные экранные снимки, инструкции и прочую информацию.
2. E-mail (Электронная почта). На Google Play может быть опубликован адрес вашей электронной почты, на который пользователи могут отправлять свои вопросы, отсылать отчеты об ошибках и информацию другого рода.
3. Phone number (Телефонный номер). Порой на Google Play публикуется телефонный номер разработчика приложений, но если вы не сможете обеспечивать телефонную

поддержку пользователей приложения, лучше ничего не указывайте в этом поле. Можете также указать телефонный номер технической поддержки на своем веб-сайте.

Согласитесь с условиями юридических документов

1. Ознакомьтесь с документом *Android Content Guidelines*, который находится на веб-сайте www.android.com/market/terms/developer-content-policy.html (см. раздел 2.4), а затем установите флажок This application meets Android Content Guidelines (Это приложение соответствует условиям Android Content Guidelines).
2. Затем нужно подтвердить, что приложение является субъектом экспортного права United States (действие этого права обычно распространяется на приложения, которые используют шифрование). Тем самым вы подтверждаете, что приложение попадает под юрисдикцию этого права и авторизовано для экспорта за пределы США. Для подтверждения установите соответствующий флажок. Для получения дополнительных сведений об экспортном праве щелкните на ссылке Learn More (Дополнительные сведения).

Как только вы будете готовы к публикации приложения, щелкните на кнопке Publish (Опубликовать). В противном случае щелкните на кнопке Save (Сохранить), чтобы сохранить информацию для публикации в дальнейшем.

2.9. Другие «рынки приложений» Android

Помимо Google Play приложения можно распространять с помощью других «рынков приложений» (табл. 2.9) или даже с помощью вашего собственного веб-сайта при участии таких служб, как AndroidLicenser (www.androidlicenser.com). В соответствии с условиями обслуживания Google Play (Terms of Service) вы не имеете право использовать информацию о заказчиках, полученную с Google Play, при продаже или распространении приложения в других местах.

Таблица 2.9. Другие «рынки приложений» Android

Рынок приложений	URL-ссылка
Amazon Appstore	developer.amazon.com/welcome.html
AndAppStore	www.andappstore.com
Androidguys	store.androidguys.com/home.asp
Andspot Market	www.andspot.com
GetJar	www.getjar.com
Handango	www.handango.com
Mplayit™	www.mplayit.com
PocketGear	www.pocketgear.com
Shop4Apps™	developer.motorola.com/shop4apps/

Рынок приложений	URL-ссылка
SlideMe	www.slideme.org
Youpark	www.youpark.com
Zeewe	www.zeewe.com

2.10. Вопросы ценообразования

Разработчик может сам формировать цены на приложения, распространяемые через Google Play. Очень часто разработчики предлагают бесплатные приложения, которые рекламируют платные версии этих же приложений с расширенным набором функций и возможностей. При этом они получают прибыль в результате продажи платных версий приложений или от рекламы, встроенной в приложение. В списке 2.6 перечислены способы монетизации приложений.

СПИСОК 2.6. СПОСОБЫ МОНЕТИЗАЦИИ ПРИЛОЖЕНИЙ

- Продажа приложения на Google Play или на других «рынках приложений» Android.
- Продажа платных обновлений для приложения.
- Продажа виртуальных товаров (см. раздел 2.12).
- Использование служб мобильной рекламы, встраиваемой в приложения (см. раздел 2.14).
- Продажа заказчикам рекламного места в приложении.
- Реклама версий приложения, обладающих расширенным набором функций.

Платные приложения

В соответствии с результатами исследований рынка приложений, производимыми аналитической фирмой Distimo (www.distimo.com/), средняя цена платного приложения Android составляет $3,626[1] (медианное значение составляет $2,727[2]). И хотя на первый взгляд эти цены кажутся небольшими, следует учитывать то, что успешные приложения могут продаваться десятками, сотнями тысяч или даже миллионами копий! Согласно сведениям компании AdMob (www.admob.com/), пользователи Android в среднем приобретают до пяти приложений в месяц[3]. Прежде чем установить цену на приложение, ознакомьтесь с ценами конкурентов. Каковы цены продаваемых ими приложений? Имеют ли эти приложения подобный набор функций? Будут ли ваши приложения более «продвинутыми»? Предлагает ли ваше приложение функции приложений конкурентов, но по более низкой цене? Хотите ли вы вернуть потраченные на разработку приложения средства и получить прибыль?

[1] gizmodo.com/5479298/android-app-store-is-57-free-compared-to-apples-25.

[2] android-apps.com/tag/median-price/.

[3] metrics.admob.com/2010/06/may-2010-mobile-metrics-report/.

Финансовые транзакции при продаже платных приложений на Google Play выполняются с помощью учетной записи Google Checkout (checkout.google.com). Пользователи услуг, предоставляемых некоторыми мобильными операторами (например, AT&T, Sprint и T-Mobile), могут воспользоваться биллинговыми системами этих операторов для взимания платы с клиентов, которые загрузили платные приложения через линии беспроводной связи. Компания Google оставляет себе 30 % от цены, по которой продается приложение, а остальные 70 % получаете вы как разработчик приложения. Торговцы с учетной записью Google Checkout получают деньги от продажи приложений ежемесячно[1]. Дополнительно вашему банку может потребоваться несколько рабочих дней, чтобы перевести платеж на ваш счет. Также придется заплатить налоги с прибыли, полученной с помощью Google Play.

Бесплатные приложения

В настоящее время количество бесплатных приложений, предназначенных для Android, существенно превышает количество приложений для iPhone[2]. Приблизительно 57 % всех приложений на Google Play предлагаются бесплатно, они же составляют большую часть всего объема загрузок[3]. Учитывая склонность большинства пользователей к загрузке бесплатных приложений, предлагайте «облегченную» версию приложения, которую могут загрузить и испытать пользователи. Например, если разработанное вами приложение представляет собой игру, предложите «облегченную версию», которая включает несколько первых уровней. После прохождения бесплатного уровня на экране появится сообщение, в котором пользователю предлагается приобрести коммерческую версию приложения, содержащую множество уровней и доступную на Google Play. Также пользователю могут предлагаться дополнительные уровни приложения с помощью встроенной в приложение биллинговой системы (для более «гладкого» обновления). Согласно результатам исследований, которые проводились компанией AdMob, необходимость обновления с «легкой» версии приложения — это одна из причин, в силу которой пользователи предпочитают платные приложения[4].

Многие компании используют бесплатные приложения для продвижения своей торговой марки, а также продвижения других продуктов и услуг (табл. 2.10).

Таблица 2.10. Бесплатные приложения Android, используемые для продвижения торговой марки

Бесплатное приложение	Набор функций
Amazon®	Просмотр и приобретение товаров на Amazon с мобильного устройства

[1] checkout.google.com/support/sell/bin/answer.py?hl=en&answer=25400.

[2] techcrunch.com/2011/04/27/there-are-now-more-free-apps-for-android-than-for-theios-platform-distimo/?utm_source=feedburner&utm_medium=email&utm_campaign=Feed%3A+Techcrunch+%28TechCrunch%29.

[3] gizmodo.com/5479298/android-app-store-is-57-free-compared-to-apples-25.

[4] metrics.admob.com/wp-content/uploads/2009/08/AdMob-Mobile-Metrics-July-09.pdf.

Бесплатное приложение	Набор функций
Bank of America	Поиск банкоматов и отделений банка в вашем районе, проверка баланса и платежей
Best Buy®	Поиск и приобретение товаров на Best Buy
Epicurious Recipe	Тысячи кулинарных рецептов, предлагаемыми журналами, издаваемыми Conde Nast, в том числе журналами *Gourmet* и *Bon Appetit*
ESPN® ScoreCenter	Настройка персонализированных табло, позволяющих отслеживать успехи ваших любимых спортивных команд (любителей и профессионалов)
Men's Health Workouts	Просмотр рекомендаций из популярных мужских журналов
NFL Mobile	Последние новости и обновления NFL, «живое» программирование, отчеты NFL и другая информация
UPS® Mobile	Отслеживание почтовых отправлений, поиск утерянных посылок, примерная оценка стоимости почтовых отправлений и другой подобной информации
NYTimes	Чтение статей из журнала *New York Times* (бесплатно или за отдельную плату)
Pocket Agent™	Приложение от State Farm Insurance, с помощью которого можно связаться с агентом, затребовать файл, найти местные отделения, проверить состояние счета в банке State Farm и совместно используемые фонды, а также выполнить ряд других операций
ING Direct ATM Finder	Поиск бесплатных банкоматов по GPS или адресу
Progressive® Insurance	Отчет о происшествии и отправка фотографий с места происшествия, поиск локального агента, получение сведений о страховке при покупке нового автомобиля и ряда других сведений
USA Today®	Чтение статей из *USA Today* и получение сведений о результатах последних спортивных соревнований
Wells Fargo®	Мобильный поиск находящихся поблизости банкоматов и отделений банка, проверка баланса, проведений платежей и оплата счетов

2.11. Монетизация приложений с помощью встроенной рекламы

Некоторые разработчики предлагают бесплатные приложения, монетизированные с помощью *встроенной в приложение рекламы*. Эта монетизация обеспечивается баннерной рекламой, подобной используемой на веб-сайтах. Мобильные рекламные сети, такие

как AdMob (www.admob.com/) и Google AdSense for Mobile (www.google.com/mobileads/publisher_home.html), агрегируют рекламные предложения от рекламодателей и отображают их в форме рекламы для приложений (см. раздел 2.15). При этом вы можете получить прибыль за рекламу на основе количества просмотров. Первая сотня наиболее популярных бесплатных приложений может принести прибыль от просмотра встроенной рекламы, сумма может варьироваться от нескольких сотен до нескольких тысяч долларов в день. Конечно, встроенная реклама не будет столь доходной для большинства приложений, поэтому если вы хотите вернуть средства, потраченные на разработку приложения и получить прибыль, продавайте приложение за деньги. Согласно данным, полученным компанией Pinch Media, 20 % пользователей, загрузивших бесплатное приложение iPhone, пользуются им в течение дня после загрузки, но лишь 5 % продолжают использовать это приложение через месяц после загрузки[1]. У нас отсутствуют подобные сведения для пользователей Android-приложений, хотя, скорее всего, они будут аналогичными. Если ваше приложение не относится к категории «суперпопулярных», прибыль от встроенной рекламы будет незначительной.

2.12. Монетизация приложений: продажа виртуальных товаров с помощью сервиса In-app Billing

С помощью встроенного в Google Play сервиса *In-app Billing* (биллинг, встроенный в приложения) можно продавать *виртуальные товары* (цифровой контент) с помощью приложений или устройств, на которых выполняется Android 2.3 или более старшая версия (табл. 2.11). Согласно данным Google, приложения, использующие встроенный биллинг, позволяют обеспечить уровень прибыли, превышающий прибыль от продажи платных приложений. Среди десятка игр, приносящих наибольшую прибыль на Google Play, девять используют встроенный биллинг[2]. Сервис In-app Billing Service доступен только для приложений, приобретенных через Google Play. Если же приложение продается через другие виртуальные магазины, этот сервис может быть недоступен. Чтобы воспользоваться встроенным биллингом, потребуется учетная запись, дающая право публикации на Google Play (см. раздел 2.4), и учетная запись продавца Google Checkout (см. раздел 2.5). Компания Google взимает 5 % от выручки за продажу приложений, другие «магазины приложений» могут оставлять себе до 30 % от выручки за продажу приложений.

Продажа виртуальных товаров может принести бóльшую прибыль (из расчета на одного пользователя), чем реклама[3]. В результате продажи виртуальных товаров в США в 2010 году было получено 1,6 млрд долларов прибыли (и 10 млрд прибыли во всем мире[4]). В 2011 году в США прибыль от продажи виртуальных товаров выросла

---

1 www.techcrunch.com/2009/02/19/pinch-media-data-shows-the-average-shelf-life-ofan-iphone-app-is-less-than-30-days/.

2 www.youtube.com/watch?v=GxU8N21wfrM.

3 www.virtualgoodsnews.com/2009/04/super-rewards-brings-virtual-currency-platformto-social-web.html.

4 www.internetretailer.com/2010/05/28/consumers-are-buying-digital-goods-new-ways.

до 2,1 млрд долларов[1]. Виртуальные товары успешно продаются на ряде веб-сайтов, в том числе на сайте Second Life®, World of Warcraft®, Farmville™ и Stardoll™. Этот вид товаров особенно популярен в мобильных играх. В соответствии с результатами, полученными исследовательской компанией Frank N. Magid Associates, из более чем 70 млн жителей США, владеющих смартфонами, 16 % тратит около $41 в год на приобретение виртуальных товаров, предназначенных для использования в играх[2].

Таблица 2.11. Виртуальные товары

Подписки на журналы	Локализованные руководства	Аватары
Виртуальные доспехи	Игровые уровни	Игровые сцены
Дополнительные функции	Рингтоны	Пиктограммы
Электронные открытки	Электронные подарки	Виртуальные деньги
Обои	Изображения	Виртуальные домашние животные
Аудиозаписи	Видеозаписи	Электронные книги

Чтобы реализовать биллинг, встроенный в приложение, выполните следующие действия:

1. В файл манифеста приложения добавьте разрешение com.android.vending.BILLING. Затем выполните действия по загрузке приложения, описанные в разделе 2.8.
2. Зарегистрируйтесь в учетной записи Google Play, дающей право публикации, на веб-сайте market.android.com/publish.
3. Перейдите к списку All Google Play Listings (Все листинги Google Play). На экране появится список загруженных приложений. Выберите подходящее приложение и щелкните на пункте In-app Products (Продукты в приложении). На экране отобразится страница, содержащаяся сведения обо всех продуктах для данного приложения.
4. *Щелкните на кнопке* Add in-app product (Добавить продукт в приложение). На экране появится страница Create New In-app Product, где можно ввести сведения о каждом продукте.
5. In-app product ID (Идентификатор продукта, встроенного в приложения). Введите идентифицирующий код (до 100 символов), используемый для каждого отдельного встроенного в приложение продукта. Идентификатор начинается с числа или строчной буквы и может включать числа, строчные буквы, символы подчеркивания (_) и точки (.).
6. Purchase type (Тип покупки). Если установить переключатель Managed per user account (Управляется учетными записями пользователями), выбранная покупка может быть приобретена только для определенной пользовательской записи. Если

[1] www.bloomberg.com/news/2010-09-28/u-s-virtual-goods-sales-to-top-2-billion-in-2011-report-says.html.

[2] www.webwire.com/ViewPressRel.asp?aId=118878.

же выбран переключатель Unmanaged (Не управляемый), пользователи могут приобретать выбранную покупку несколько раз.

7. Publishing state (Состояние публикации). Чтобы открыть доступ пользователей к продуктам, параметру publishing state присвойте значение Published (Опубликовано).
8. Language (Язык). По умолчанию язык продукта совпадает с языком, выбранным при загрузке и публикации продукта.
9. Title (Заголовок). Уникальный заголовок продукта (до 25 символов), который отображается для пользователей.
10. Description (Описание). Краткое описание (до 80 символов) продукта, которое отображается для пользователей.
11. Price (Цена). Цена продукта, выраженная в долларах США.
12. Щелкните на кнопке Publish (Опубликовать), чтобы продукты были доступны пользователям, либо на кнопке Save (Сохранить), если продукт будет опубликован позже.

Чтобы получить дополнительные сведения о биллинге, встроенном в приложения, включая примеры приложений, описания лучших методик и другую информацию, посетите веб-сайт developer.android.com/guide/market/billing/index.html.

Покупки с помощью встроенного в приложения биллинга через альтернативные «рынки приложений»

Если вы намереваетесь продавать приложения с помощью других «рынков приложений» (см. раздел 2.9), обратитесь к независимым провайдерам мобильных платежей. В результате у вас появится возможность встроить в приложения биллинг, реализованный с помощью соответствующих библиотек API (табл. 2.12). Начните с создания дополнительных заблокированных функциональных свойств (например, игровых уровней или аватаров). Если пользователь планирует что-либо приобрести, инструмент приобретения, встроенный в приложение, обрабатывает финансовую транзакцию и передает сообщение приложению, которое верифицирует платеж. Затем приложение выполняет разблокировку дополнительных функций. Согласно сведениям компании Boku, которая занимается мобильными платежами, провайдеры мобильной связи забирают себе от 25 до 45 % от стоимости приложения[1].

Таблица 2.12. Провайдеры мобильных платежей, используемых при покупках с помощью встроенной в приложения системы биллинга

Провайдер	URL-ссылка	Описание
PayPal Mobile Payments Library	www.x.com/community/ppx/xspaces/mobile/mep	Пользователи щелкают на кнопке Pay with PayPal (Оплатить с помощью PayPal), регистрируются в своих учетных записях PayPal, а затем щелкают на кнопке Pay (Оплатить)

[1] www.boku.com/help/faq/publisher/.

Провайдер	URL-ссылка	Описание
Zong	www.zong.com/android	Отображает кнопку Buy (Купить) для платежей, выполняемых с помощью единственного щелчка мыши. Платежи отображаются в счете телефонных услуг, получаемых пользователем
Boku	www.boku.com	После щелчка на кнопке Pay by Mobile (Оплатить через мобильную связь) введите номер мобильного телефона, а затем завершите транзакцию, ответив на текстовое сообщение, полученное на мобильный телефон

2.13. Запуск приложения Market из пользовательского приложения

Чтобы дополнительно стимулировать продажи приложений, предоставьте пользователю возможность запускать приложение Market (Google Play) непосредственно из пользовательского приложения (обычно с помощью соответствующей кнопки, отображаемой на экране приложения). В результате пользователь получит возможность загружать другие опубликованные вами приложения либо приобрести связанное приложение, которое имеет набор функций, отличающихся от функций ранее загруженной версии приложения. Можно также предоставить пользователю возможность загрузки последних обновлений приложения путем запуска приложения Market.

Существует два способа использования приложения Market. Во-первых, можно отобразить на экране Google Play результаты поиска приложений в соответствии с такими категориями, как имя разработчика, имя пакета или строка символов. Например, если нужно стимулировать пользователей загружать другие ваши приложения, опубликованные на Google Play, включите соответствующую кнопку, отображаемую на экране приложения. Как только пользователь нажмет эту кнопку, запустится приложение Market, которое инициализирует поиск приложений, включающих ваше имя или название компании. Второй способ использования приложения Market — отобразить страницу сведений о соответствующем приложении на экране приложения Market.

Дополнительные сведения о методике, используемой для запуска приложения Market из другого приложения, изложены в документе *Publishing Your Applications: Using Intents to Launch the Market Application on a Device*, который можно загрузить с веб-сайта developer.android.com/guide/publishing/publishing.html#marketintent.

2.14. Управление приложениями, находящимися на Google Play

С помощью консоли разработчика Google Play Developer Console можно управлять учетной записью и приложениями, проверять пользовательский рейтинг вашего приложения (от 0 до 5 звездочек), отслеживать количество общих и активных инсталляций приложения (разница между количеством инсталляций и удалений приложения).

Можно также ознакомиться с тенденциями инсталляций и загрузок копий приложения для различных версий Android, устройств и прочей информацией. Путем создания отчетов об ошибках приложений Android (Android Application Error Reports) накапливается и «замораживается» информация об ошибках, пересылаемая пользователями. Если вы разработали обновление приложения, то с минимальными усилиями можете опубликовать новую версию. Можно также удалить приложение из Market, но если пользователи заблаговременно загрузили его, оно останется на пользовательских устройствах. Пользователи, которые удалили приложение, могут установить его повторно (это приложение остается на серверах Google до тех пор, пока не будет удалено за нарушение правил использования, изложенных в документе *Terms of Service*).

2.15. Маркетинг приложения

После публикации приложения представьте его аудитории будущих пользователей[1]. И в этом вам поможет вирусный маркетинг (распространение слухов) с помощью сайтов социальных сетей, таких как Facebook, Twitter и YouTube. Эти сайты имеют огромную аудиторию. Согласно сведениям comScore, аудитория YouTube насчитывает до 10 % от общего числа пользователей Интернета, а Facebook — достигает 17 %[2]. В табл. 2.13 приведены сведения о наиболее популярных сайтах социальных сетей. Зачастую в качестве недорогих и эффективных средств маркетинга используются рассылки по электронной почте и электронные бюллетени новостей.

Таблица 2.13. Популярные веб-сайты социальных сетей

Веб-сайт социальной сети	URL-ссылка	Описание
Facebook	www.facebook.com	Социальные сети
Twitter	www.twitter.com	Микроблоги, социальные сети
Groupon	www.groupon.com	Социальная коммерция
Foursquare	www.foursquare.com	Скидки
Gowalla	www.gowalla.com	Скидки
YouTube	www.youtube.com	Публикация видеороликов в Интернете
LinkedIn	www.linkedin.com	Социальные сети для бизнеса
Flickr	www.flickr.com	Публикация фотографий в Интернете
Digg	www.digg.com	Общий доступ и обнаружение контента
StumbleUpon	www.stumbleupon.com	Социальный букмаркинг
Delicious	www.delicious.com	Социальный букмаркинг

[1] Чтобы получить дополнительные сведения о маркетинге Android-приложений, обратитесь к книге *Android Apps Marketing: Secrets to Selling Your Android App*, написанной Джеффри Хаггисом (Jeffrey Hughes).

[2] tech.fortune.cnn.com/2010/07/29/google-the-search-party-is-over/.

Веб-сайт социальной сети	URL-ссылка	Описание
Bebo	www.bebo.com	Социальные сети
Tip'd	www.tipd.com	Социальные новости из мира финансов и бизнеса
Blogger	www.blogger.com	Блоггинг сайтов
Wordpress	www.wordpress.com	Блоггинг сайтов
Squidoo	www.squidoo.com	Платформа, используемая для публикаций и создания сообществ

Фейсбук

Один из наиболее популярных сайтов социальных сетей, Фейсбук, имеет более 600 млн активных пользователей (более 200 млн в начале 2009 года[1]). Каждый из пользователей имеет в среднем 130 друзей[2], причем количество пользователей возрастает примерно на 5 % в месяц! Это превосходный ресурс для вирусного маркетинга (путем распространения слухов).

Начните с настройки официальной страницы Фейсбука для вашего приложения. Используйте эту страницу для:

- размещения информации о приложении;
- публикации новостей;
- доступа к обновлениям;
- публикации обзоров;
- размещения советов;
- публикации видеороликов;
- публикации экранных снимков;
- отображения таблицы рекордов игры;
- осуществления обратной связи с пользователем;
- размещения ссылок на Google Play, откуда пользователи могут загрузить приложение.

Теперь, когда вы наслышаны о популярности Фейсбука, приступайте к распространению слухов. Попросите ваших коллег и друзей «полюбить» вашу страницу на Фейсбуке и посоветовать своим друзьям и знакомым сделать то же самое. По мере того как пользователи будут посещать вашу страницу, заметки о впечатлениях будут появляться на страницах новостей друзей этих пользователей, способствуя росту «армии» поклонников вашего приложения.

[1] topics.nytimes.com/top/news/business/companies/facebook_inc/index.html.

[2] techcrunch.com/2010/07/15/facebook-500-million-users/?utm_source=feedburner&utm_medium=email&utm_campaign=Feed:+Techcrunch+(TechCrunch).

Твиттер

Твиттер — это сайт микроблогов и социальных сетей, который каждый месяц посещает более 190 млн пользователей[1]. Пользователи Твиттера обмениваются *твитами* — сообщениями длиной до 140 символов. Твиттер рассылает твиты всем последователям (на время написания книги одна известная рок-звезда имела более 8,5 млн последователей). Многие пользователи используют Твиттер для отслеживания новостей и тенденций. Поэтому можно рассылать твиты, содержащие сведения о приложении (в том числе анонсы новых версий, советы, факты, комментарии для пользователей и другую информацию). Также посоветуйте друзьям и коллегам разослать твиты, содержащие сведения о приложении. Воспользуйтесь *хеш-тегом* (#) для создания ссылки на сообщение. Например, в процессе рассылки твитов, содержащих информацию о нашей книге, в нашей подписке на новости Twitter, @deitel мы используем хеш-тег #AndroidFP. Другие пользователи могут воспользоваться этим хеш-тегом, чтобы добавить комментарии о книге. Это облегчит поиск твистов для сообщений, связанных с книгой *Android for Programmers*.

Вирусные видеоролики

Еще один способ распространения сведений о приложении — воспользоваться вирусными видеороликами, опубликованными на сайтах публикации видеороликов в Интернете (например, YouTube, Dailymotion, Bing Videos, Yahoo! Video), сайтах социальных сетей (например, Фейсбук, Твиттер, MySpace), путем рассылки сообщений по электронной почте. Если вы создадите поистине неотразимый видеоролик, который станет предметом обсуждения среди пользователей, это будет способствовать быстрому росту популярности вашего приложения среди пользователей социальных сетей.

Бюллетени новостей, рассылаемые по электронной почте

Для продвижения приложения можно воспользоваться бюллетенями новостей, которые рассылаются по электронной почте. Добавьте ссылки на Google Play, с помощью которых пользователи смогут загрузить приложение. Также добавьте ссылки на страницы в социальных сетях (например, на страницу Фейсбука или на подписку Твиттера). Благодаря этому пользователи могут быть в курсе последних новостей, касающихся вашего приложения.

Обзоры приложений

Свяжитесь с сайтами, на которых ведутся специализированные блоги и помещаются обзоры приложений (табл. 2.14), и сообщите сведения о своем приложении. Сообщите им промо-код для бесплатной загрузки приложения (см. раздел 2.10). Учтите, что подобные сайты получают большое количество запросов, поэтому будьте кратки и предельно информативны. Многие обозреватели приложений публикуют видеообзоры приложений на YouTube и других подобных сайтах (табл. 2.15).

[1] techcrunch.com/2010/06/08/twitter-190-million-users/.

Таблица 2.14. Веб-сайты, публикующие обзоры Android-приложений

Название веб-сайта	URL-ссылка
Android Tapp™	www.androidtapp.com/
Appolicious™	www.androidapps.com
AppBrain	www.appbrain.com
Best Android Apps Review	www.bestandroidappsreview.com
AppStoreHQ	android.appstorehq.com
Android App Review Source	www.androidappreviewsource.com
Androinica	www.androinica.com
AndroidZoom	www.androidzoom.com
AndroidLib	www.androlib.com
Android and Me	www.androidandme.com
AndroidGuys	www.androidguys.com/category/reviews/
Android Police	www.androidpolice.com/
Phandroid	www.phandroid.com

Таблица 2.15. Примеры сайтов, на которых публикуются учебные видеоролики Android-приложений

Пример веб-сайта	URL-ссылка
ADW Launcher	www.youtube.com/watch?v=u5gRgpuQE_k
Daily App Show	dailyappshow.com
Timeriffic	androidandme.com/2010/03/news/android-app-video-rseview-timeriffic/
Frackulous	frackulous.com/141-glympse-android-app-review/
Moto X Mayhem	www.appvee.com/games/articles/6968-android-appvideo-review-moto-x-mayhem

Связи с интернет-общественностью

В индустрии связей с общественностью применяются средства массовой информации, предназначенные для облегчения передачи сообщений от компаний конечным пользователям. С появлением феномена, известного под названием Web 2.0, практики, занимающиеся связями с общественностью, включают блоги, подкасты, RSS-подписки и социальные медиасредства в свои PR-компании. В табл. 2.16 перечислены некоторые платные и бесплатные интернет-ресурсы, посвященные связям с общественностью, включая файлы распространения пресс-релизов, службы создания пресс-релизов и другие ресурсы. Чтобы получить сведения о дополнительных ресурсах, обратитесь к ресурсу *Internet Public Relations Resource Center* на веб-сайте www.deitel.com/InternetPR/.

Таблица 2.16. Ресурсы, обеспечивающие связи с интернет-общественностью

Интернет-ресурс	URL-ссылка	Описание
	Бесплатные ресурсы	
ClickPress™	www.clickpress.com	Отправьте на этот сайт свои новости для одобрения (бесплатно или на платной основе). Одобренные новости будут доступны на сайте ClickPress, а также на сайтах новостей
PRLog	www.prlog.org/pub/	Бесплатная подписка на пресс-релизы и рассылки
i-Newswire	www.i-newswire.com	Бесплатная подписка на пресс-релизы и рассылки
openPR®	www.openpr.com	Свободно доступные публикации в форме пресс-релизов
	Платные ресурсы	
PR Leap	www.prleap.com	Платная служба рассылки пресс-релизов в Интернете
Marketwire	www.marketwire.com	Платная служба рассылки пресс-релизов, которая позволяет выбирать целевую аудиторию по различным признакам: географический, производственный и т. д.
InternetNews-Bureau.com®	www.internetnewsbureau.com	Службы рассылки пресс-релизов в Интернете, предназначенные для бизнесменов и журналистов
Mobility PR	www.mobilitypr.com	Службы, поддерживающие связи с общественностью компаний, работающих в индустрии мобильных устройств
Press Release Writing	www.press-releasewriting.com	Службы рассылки пресс-релизов, которые предлагают ряд дополнительных услуг, такие как чтение, проверка на наличие ошибок и редактирование пресс-релизов. Ознакомьтесь с советами по написанию эффективных пресс-релизов

Мобильные рекламные сети

Приобретение рекламы (например, в других приложениях, в Интернете, в газетах и журналах либо на радио и телевидении) — это еще один способ маркетинга приложений. Мобильные рекламные сети (табл. 2.17) специализируются на рекламе мобильных приложений, разработанных для платформы Android (и других платформ). Воспользуйтесь этими сетями (на платной основе) для маркетинга разработанных вами приложений Android. Имейте в виду, что большинство приложений практически не приносят прибыль, поэтому не тратьте много денег на рекламу. Рекламные сети можно также использовать для монетизации свободно распространяемых приложений путем

включения в их состав баннеров. Многие из мобильных рекламных сетей могут выбирать целевую аудиторию по местоположению, провайдеру, устройству (например, Android, iPhone, BlackBerry и другие устройства) и другим категориям.

Таблица 2.17. Мобильные рекламные сети

Мобильная рекламная сеть	URL-ссылка	Описание
AdMob	www.admob.com/	Реклама приложения в Интернете и в других приложениях либо включение рекламы в приложение в целях монетизации
Google AdSense for Mobile	www.google.com/mobileads/	Отображение рекламы Google (предназначенной для мобильных платформ) в составе мобильных приложений или мобильных веб-страниц. Рекламодатели также могут размещать рекламу на мобильной версии YouTube
AdWhirl (by AdMob)	www.adwhirl.com	Сервис с открытым исходным кодом, который агрегирует несколько сетей мобильной рекламы, обеспечивая увеличение рейтинга заполнения рекламы (частота, с которой реклама появляется в приложении)
Medialets	www.medialets.com	Набор инструментов SDK мобильной рекламы обеспечивает включение рекламы в приложение. Аналитический набор SDK обеспечивает отслеживание использования приложения и рейтинг щелчков на рекламных баннерах
Nexage	www.nexage.com	Набор SDK мобильной рекламы позволяет включать рекламу в ваше приложение, заимствованную из многочисленных рекламных сетей, а затем управлять рекламой с помощью единой приборной панели генерирования отчетов
Smaato®	www.smaato.net	Платформа оптимизации рекламы Smaato's SOMA (Smaato Open Mobile Advertising) агрегирует более 50 мобильных рекламных сетей
Decktrade™	www.decktrade.com	Реклама приложения на мобильных сайтах либо включение рекламы в приложение в целях монетизации
Flurry™	www.flurry.com/	Аналитические инструменты, применяемые для отслеживания загрузок, использования приложений Android и подсчета полученной прибыли

Средства, выделяемые на рекламу

Показатель *eCPM* (effective cost per 1000 impressions, эффективные средства на 1000 впечатлений) для рекламы приложений Android изменяется от 0,09 до 4 долларов,

в зависимости от рекламной сети и рекламы[1]. Большая часть платежей за рекламу на платформе Android основана на рейтинге щелчков (CTR, clickthrough rate), а не на количестве генерируемых впечатлений. Если показатели CTR для рекламы в вашем приложении высоки, ваша рекламная сеть может использоваться для высоко оплачиваемой рекламы, увеличивая вашу прибыль. Величины показателей CTR изменяются от 1 до 2 % от количества рекламы в приложениях (в зависимости от используемого приложения).

2.16. Другие популярные платформы приложений

Чтобы существенно увеличить количество пользователей приложения, подумайте о переносе Android-приложений на другие платформы, такие как iPhone и BlackBerry (табл. 2.18). Согласно результатам исследований компании AdMob, более 70 % разработчиков iPhone-приложений планируют разработать версию приложения для Android на протяжении последующих шести месяцев, а 48 % разработчиков Android-приложений планируют перенести их на платформу iPhone[2]. Различия объясняются тем, что приложения iPhone разрабатываются на компьютерах Macintosh, которые стоят дороже. К тому же для разработки применяется язык программирования Objective-C, который знаком небольшому числу разработчиков. Приложения Android могут разрабатываться на компьютерах Windows, Linux или Macintosh с помощью языка программирования Java, который является одним из наиболее широко используемых языков программирования в мире. Новейший планшет BlackBerry Playbook также может выполнять приложения Android (доступен в виртуальном магазине BlackBerry App World store).

Таблица 2.18. Другие популярные платформы для приложений (помимо Android)

Платформа	URL-ссылка
	Платформы мобильных приложений
BlackBerry (RIM)	na.blackberry.com/eng/services/appworld/?
iOS (Apple)	developer.apple.com/iphone/
webOS (Palm)	developer.palm.com
Windows Phone 7	developer.windowsphone.com
Symbian	developer.symbian.org
	Платформы интернет-приложений
Facebook	developers.facebook.com
Twitter	apiwiki.twitter.com
Foursquare	developer.foursquare.com

[1] seoamit.wordpress.com/2010/02/06/monetizing-mobile-apps-android-and-iphone/.

[2] metrics.admob.com/wp-content/uploads/2010/03/AdMob-Mobile-Metrics-Mar-10-Publisher-Survey.pdf.

Платформа	URL-ссылка
Gowalla	gowalla.com/api/docs
Google	code.google.com
Yahoo!	developer.yahoo.com
Bing	www.bing.com/developers
Chrome	code.google.com/chromium/
LinkedIn	developer.linkedin.com/index.jspa

2.17. Документация для Android-разработчиков

В табл. 2.19 перечислены некоторые из ключевых документов, предназначенных для Android-разработчиков. Дополнительную документацию можно найти на веб-сайте developer.android.com/.

Таблица 2.19. Документация, предназначенная для Android-разработчиков

Документ	URL-ссылка
Application Fundamentals	developer.android.com/guide/topics/fundamentals.html
Manifest.permission Summary	developer.android.com/reference/android/Manifest.permission.html
AndroidManifest.xml File <usesfeature> *Element*	developer.android.com/guide/topics/manifest/uses-feature-element.html
Android Compatibility	developer.android.com/guide/practices/compatibility.html
Supporting Multiple Screens	developer.android.com/guide/practices/screens_support.html
Designing for Performance	developer.android.com/guide/practices/design/performance.html
Designing for Responsiveness	developer.android.com/guide/practices/design/responsiveness.html
Designing for Seamlessness	developer.android.com/guide/practices/design/seamlessness.html
Android User Interface Guidelines	developer.android.com/guide/practices/ui_guidelines/index.html
Icon Design Guidelines	developer.android.com/guide/practices/ui_guidelines/icon_design.html
Android Market Content Policy for Developers	www.android.com/market/terms/developer-content-policy.html
In-app Billing	developer.android.com/guide/market/billing/index.html
Android Emulator	developer.android.com/guide/developing/tools/emulator.html

продолжение ➪

Таблица 2.19 *(продолжение)*

Документ	URL-ссылка
Versioning Your Applications	developer.android.com/guide/publishing/versioning.html
Preparing to Publish: A Checklist	developer.android.com/guide/publishing/preparing.html
Market Filters	developer.android.com/guide/appendix/market-filters.html
Localization	developer.android.com/guide/topics/resources/localization.html
Technical Articles	developer.android.com/resources/articles/index.html
Sample Apps	developer.android.com/resources/samples/index.html
Android FAQs	developer.android.com/resources/faq/index.html
Common Tasks and How to Do Them in Android	developer.android.com/resources/faq/commontasks.html
Using Text-to-Speech	developer.android.com/resources/articles/tts.html
Speech Input	developer.android.com/resources/articles/speech-input.html

2.18. Шутим вместе с Android

В табл. 2.20 перечислены веб-сайты, на которых можно найти шутки, связанные с Android.

Таблица 2.20. Android-юмор

Веб-сайт	Описание
crenk.com/android-vs-iphone-humor/	Забавная картинка, иллюстрирующая ключевые отличия Android от iPhone
www.collegehumor.com/video:1925037	Юмористический видеоролик от CollegeHumor, который убеждает вас приобрести смартфон на базе Android
www.youtube.com/watch?v=MAHwDx0lI-M	www.youtube.com/watch?v=MAHwDx0lI-M Юмористический видеоролик, «Samsung Behold II Man Adventures — Part 1.»
www.theonion.com/video/new-googlephone-service-whispers-targetedads-dir, 17470/	Видеоролик от Onion, «New Google Phone Service Whispers Targeted Ads Directly in Users' Ears.»
www.collegehumor.com/article:1762453	Статья «A Few Problems with the New Google Phone» от CollegeHumor, подшутившим над функцией «Did-You-Mean» от Google Search.

2.19. Резюме

В этой главе был рассмотрен процесс регистрации на Google Play и настройки учетной записи Google Checkout, обеспечивающей продажу приложений. Продемонстрирована

подготовка приложений к публикации на Google Play, включая тестирование на эмуляторе и устройствах Android, создание пиктограмм и всплывающих экранов. Рассмотрены лучшие методики разработки приложений и документ *Android User Interface Guidelines* и методы редактирования файла *AndroidManifest.xml*е. Уделено внимание загрузке приложений на Google Play. Рассмотрены альтернативные «рынки приложений», с помощью которых можно продавать приложения. Вашему вниманию были предложены советы по формированию цен на приложения. Указаны ресурсы, где описана монетизация приложений с помощью встроенной в них рекламы и продажи виртуальных товаров с помощью приложений. Также описаны ресурсы с различными методиками маркетинга приложений, некоторые из них доступны на Google Play.

В главе 3 демонстрируется использование интегрированной среды разработки Eclipse для создания первого Android-приложения с помощью визуального программирования без написания кода. Здесь же вы ознакомитесь с обширными функциями справки Eclipse. В главе 4 вы начнете программировать Android-приложения с помощью Java.

3 Приложение Welcome

Знакомимся с Eclipse и модулем ADT Plugin

В этой главе...

- Изучение основ интегрированной среды разработки Eclipse, применяемой для написания, выполнения и отладки Android-приложений.
- Создание проекта Eclipse, предназначенного для разработки нового приложения.
- Визуальная разработка графического интерфейса пользователя (без программирования) с помощью визуального редактора макета ADT Visual Layout Editor.
- Изменение свойств компонентов графического интерфейса пользователя.
- Создание простого приложения Android и его выполнение на виртуальном устройстве Android Virtual Device (AVD)

3.1. Введение

В этой главе мы создадим приложение Welcome. Это простое приложение отображает приветственное сообщение и два изображения, и для его создания мы не написали ни одной строчки кода. Воспользуемся возможностями интегрированной среды разработки Eclipse и модулем ADT — двумя наиболее популярными инструментами, применяемыми для разработки и тестирования Android-приложений. Мы рассмотрим возможности среды Eclipse и продемонстрируем, каким образом создается приложение Android (рис. 3.1) с помощью визуального макетного редактора ADT. Этот редактор позволяет создавать графический интерфейс пользователя с помощью технологий перетаскивания. И наконец, мы запустим приложение на виртуальном устройстве Android Virtual Device (AVD).

3.2. Обзор применяемых технологий

В этой главе вас ожидает знакомство с интегрированной средой разработки Eclipse и подключаемым модулем ADT Plugin. Вы научитесь ориентироваться в Eclipse и создавать

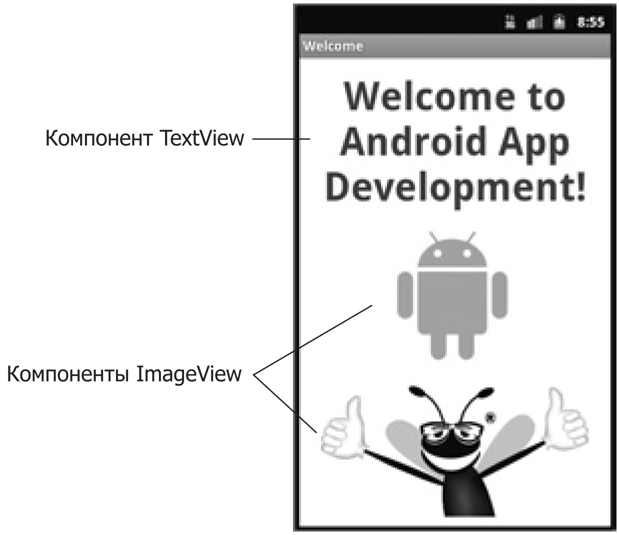


Рис. 3.1. Приложение Welcome

новые проекты. Используя визуальный макетный редактор ADT Visual Layout Editor, можно отображать картинки с помощью компонентов ImageViews и текст — с помощью компонента TextView. Будет рассмотрено редактирование свойств компонента GUI (свойство Text компонента TextView и свойства Src компонента ImageView), доступных на вкладке Eclipse Properties (Свойства), а также выполнение приложения на виртуальном устройстве Android Virtual Device (AVD).

3.3. Интегрированная среда разработки Eclipse

Приведенные в книге примеры разработаны с помощью последних (на момент написания книги) версий Android SDK (версии 2.3.3 и 3.0), а также с помощью среды разработки Eclipse IDE вместе с подключаемым модулем ADT (Android Development Tools) Plugin. Предполагается, что к началу чтения главы вы уже установили Java SE Development Kit (JDK), Android SDK и Eclipse. Установка этих компонентов была рассмотрена в разделе «Подготовительные действия» введения книги.

Введение в Eclipse

В интегрированной среде разработки Eclipse можно управлять, редактировать, компилировать, выполнять и отлаживать приложения. Подключаемый модуль ADT Plugin для Eclipse предоставляет пользователям дополнительные инструменты, применяемые для разработки Android-приложений. Можно также воспользоваться подключаемым модулем ADT Plugin для управления различными версиями платформы Android. Это требуется в случае разработки приложений, предназначенных для различных устройств, на которых установлены различные версии операционной системы Android. После первого запуска среды Eclipse появится вкладка Welcome (Добро пожаловать), которая показана на рис. 3.2. На этой вкладке находится ряд ссылок-пиктограмм, представленных в табл. 3.1. Щелкните на кнопке Workbench (Рабочая среда), чтобы отобразить

перспективу разработки Java, в которой начинается разработка Android-приложений. В Eclipse поддерживается разработка приложений на нескольких языках программирования. Каждый установленный набор инструментов Eclipse представлен отдельной перспективой разработки. В результате изменения перспективы реконфигурируются. Изменение перспектив приведет к реконфигурированию интегрированной среды разработки таким образом, чтобы можно было использовать инструменты для соответствующего языка программирования.



Рис. 3.2. Вкладка Welcome to the Eclipse IDE for Java Developers в окне Eclipse

Таблица 3.1. Ссылки, находящиеся на вкладке Welcome среды Eclipse

Ссылка	Описание
Overview (Обзор)	Обзор интегрированной среды разработки и ее функций
What's New (Что нового)	Сведения о новых качествах установленной версии Eclipse, ссылки на сообщества пользователей Eclipse в Интернете и обновления интегрированной среды разработки (DE)
Samples (Примеры)	Ссылки на примеры для загруженной конфигурации Eclipse
Tutorials (Учебники)	Учебники, которые помогут вам начать разработку Java-приложений в среде Eclipse и использовать различные возможности Eclipse
Workbench (Рабочая среда)	Переход к перспективе разработки

3.4. Создание нового проекта

Чтобы начать программировать на Android в Eclipse, выполните команды File ▸ New ▸ Project… (Файл ▸ Новый ▸ Проект…) для отображения диалогового окна New Project (Новый проект). Раскройте узел Android, выберите параметр Android Project (Проект Android) и щелкните на кнопке Next> (Далее>). На экране появится диалоговое окно New Android Project (Новый проект Android), показанное на рис. 3.3. Это окно также можно отобразить с помощью раскрывающегося списка New (Создать) панели инструментов. После создания первого проекта в меню File ▸ New и в раскрывающемся меню New появится параметр Android Project (Проект Android).

Проект — это группа связанных файлов, например файлы кода и изображения, которые формируют приложение. В диалоговом окне New Android Project можно создать

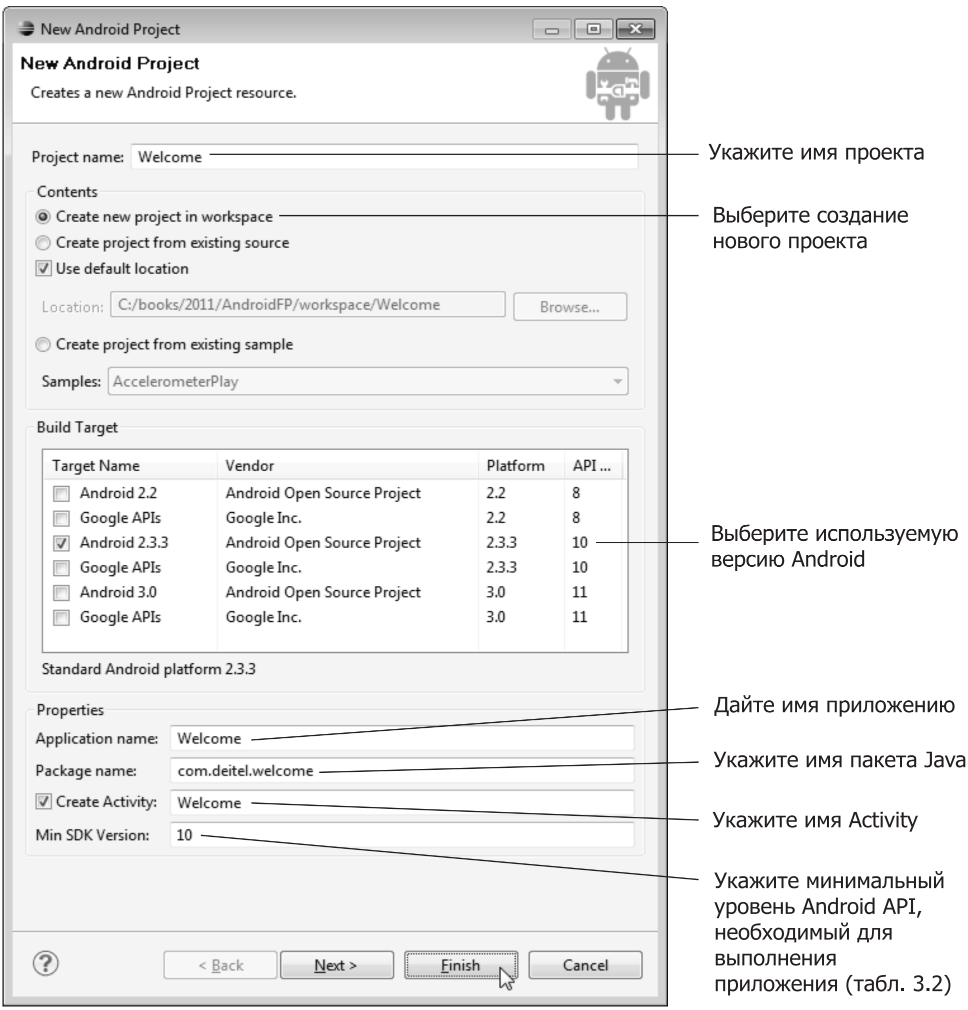


Рис. 3.3. Диалоговое окно New Android Project

проект «с нуля» либо воспользоваться существующим исходным кодом (например, примерами кода из книги).

В этом диалоговом окне укажите следующие сведения:

1. В поле Project name: (Название проекта:) ведите название Welcome. Это имя корневого узла проекта на вкладке Package Explorer (Обозреватель пакетов) в среде Eclipse.
2. В разделе Contents (Содержимое) установите флажок Create new project in workspace (Создать новый проект в рабочей области), если нужно создать новый проект «с нуля», или флажок Create project from existing source (Создать проект на основе существующего), чтобы создать новый проект на основе существующих файлов исходного кода Java.
3. В разделе Build Target (Цель построения) выберите используемую версию Android. В большинстве рассмотренных в книге примеров используется версия 2.3.3, рекомендуется выбрать минимальную версию, поддерживаемую приложением, чтобы оно могло выполняться на возможно большем числе устройств.

В разделе Properties (Свойства) этого диалогового окна введите следующую информацию:

1. В поле Application name: (Имя приложения:) введите Welcome. Мы обычно даем приложениям те же названия, что и проектам, но этого делать не обязательно. Это название отображается на панели, находящейся в верхней части приложения, если она явно не *скрыта* приложением.
2. В Android используются традиционные для Java соглашения о наименовании пакетов, предусматривающие, что имя должно состоять как минимум из двух частей (например, com.deitel). В поле Package name: (Имя пакета:) введите имя com.deitel.welcome. Мы используем наш домен deitel.com (в обратном порядке), за которым следует название приложения. Все классы и интерфейсы, которые создаются в качестве части приложения, будут помещены в этот пакет Java. Как Android, так и Android Market используют название пакета в качестве уникального идентификатора приложения.
3. В поле Create Activity: (Создать деятельность:) введите Welcome. Это имя класса, который управляет выполнением приложения. Начиная со следующей главы, мы будем изменять класс, чтобы реализовать функциональные свойства приложения.
4. В поле Min SDK Version: (Минимальная версия SDK:) укажите минимальный уровень API, который требуется для выполнения приложения. В результате приложение будет выполняться на устройствах с такой же версией API либо более старшей. В этой книге мы обычно используем API уровня 10, который соответствует Android 2.3.3, либо API уровня 11, который соответствует Android 3.0. Чтобы выполнить приложение на платформе Android 2.2 или более старшей, выберите API уровня 8. В *подобных случаях следует убедиться в том, что приложение не использует функции, которые присущи более современным версиям Android*. В табл. 3.2 приведен список версий Android SDK и уровней API. *Другие версии SDK не рекомендуются к применению и не должны использоваться*. На веб-странице developer.android.com/resources/dashboard/platform-versions.html приведена текущая доля устройств Android, работающая под соответствующей версией платформы.
5. Чтобы завершить создание проекта, щелкните на кнопке Finish (Готово).

Таблица 3.2. Версии Android SDK и уровни API (developer.android.com/sdk/index.html)

Версия Android SDK	Уровень API
3.0	11
2.3.3	10
2.2	8
2.1	7
1.6	4
1.5	3

ПРИМЕЧАНИЕ

Возможно, в процессе загрузки Android SDK вы увидите сообщения об ошибках проекта.

Окно Package Explorer

Как только создается (или открывается) проект, в левой части интегрированной среды разработки появляется окно Package Explorer (Обозреватель пакетов), обеспечивающее доступ ко всем файлам проекта. На рис. 3.4 показано содержимое проекта для приложения Welcome. В узле Welcome представлен проект. Можно одновременно открыть несколько проектов в окне интегрированной среды разработки (IDE). Каждый из проектов имеет собственный узел верхнего уровня.

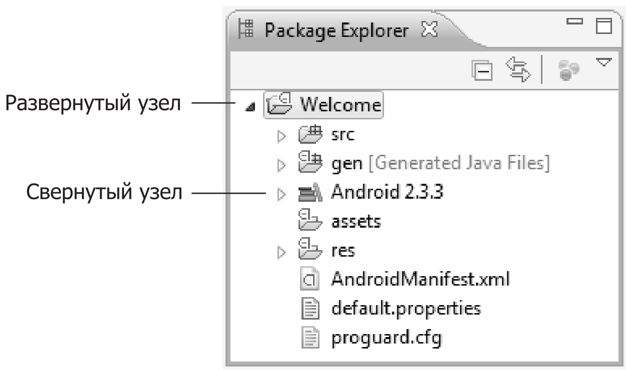


Рис. 3.4. Окно Package Explorer

В узле проекта содержимое проекта организовано в виде различных файлов и папок, в том числе:

- src — папка, включающая исходные файлы проекта Java;
- gen — папка, содержащая файлы Java, сгенерированные IDE;
- Android 2.3.3 — папка, включающая версию Android framework, выбранную при создании приложения;

- res — папка, в которой находятся *файлы ресурсов*, связанные с приложением, такие как макеты GUI и изображения, используемые в приложении.

Иные файлы и папки будут рассмотрены в других главах книги.

3.5. Создание графического интерфейса пользователя приложения Welcome с помощью визуального макетного редактора модуля ADT

А теперь создадим графический интерфейс пользователя для приложения Welcome. С помощью редактора ADT *Visual Layout Editor* можно создать графический интерфейс пользователя путем перетаскивания в окно приложения компонентов GUI, таких как Buttons, TextViews, ImageViews и др. При создании Android-приложения с помощью Eclipse *макет интерфейса пользователя хранится в XML-файле, который по умолчанию называется* main.xml. Благодаря определению элементов GUI в XML-файле можно легко отделить логику приложения от его презентации. Файлы макетов являются *ресурсами* приложения и хранятся в папке res проекта. Макеты GUI находятся в подпапке папки layout. После двойного щелчка на файле main.xml, находящегося в папке приложения /res/layout, появится представление Visual Layout Editor, выбранное по умолчанию (рис. 3.5). Чтобы просмотреть XML-содержимое файла (рис. 3.6), выберите вкладку с именем файла макета (в данном случае, main.xml). Чтобы вернуться обратно к представлению Visual Layout Editor, выберите вкладку Graphical Layout (Графический макет). Структура XML-файла будет рассмотрена в разделе 3.6.

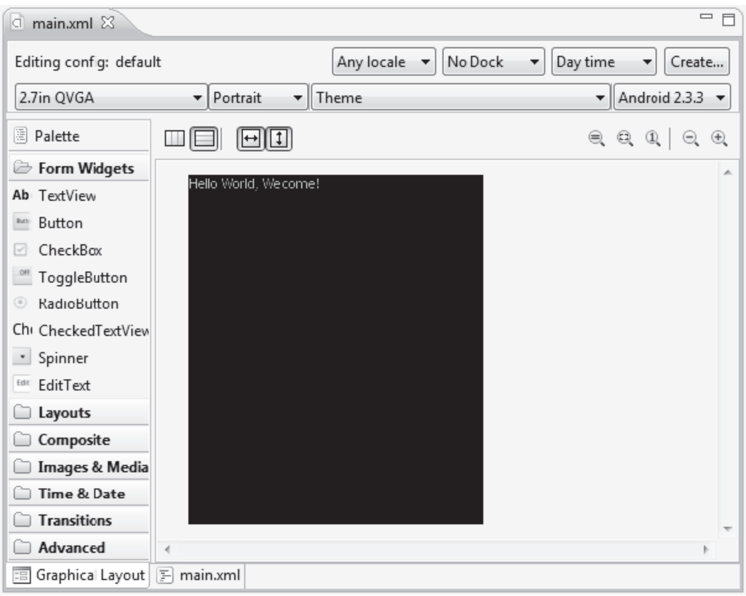
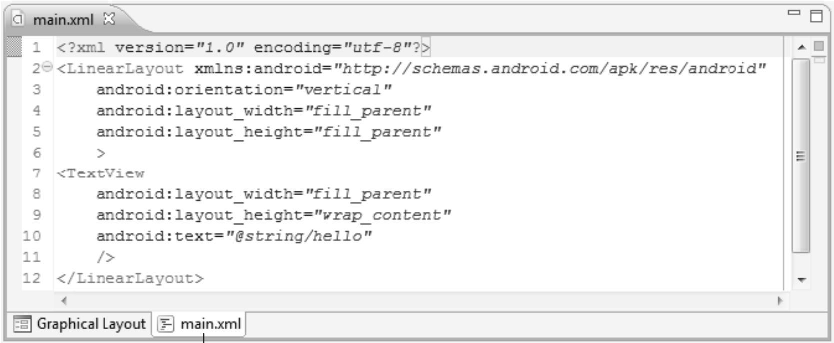


Рис. 3.5. Представление Visual Layout Editor заданного по умолчанию интерфейса приложения



Вкладка XML view

Рис. 3.6. XML-представление заданного по умолчанию GUI.приложения

Графический интерфейс пользователя, заданный по умолчанию

Графический интерфейс пользователя, заданный по умолчанию для нового Android-приложения, включает компонент LinearLayout. Этот компонент имеет черный фон и включает компонент TextView, содержащий текст "Hello World, Welcome!" (см. рис. 3.5). Компонент LinearLayout упорядочивает элементы GUI вдоль линии по вертикали или горизонтали. Компонент TextView позволяет отображать текст. Если запустить приложение на устройстве AVD или на реальном устройстве, появится заданный по умолчанию черный фон и белый текст на нем.

В табл. 3.3 перечислены некоторые макеты, определяемые пакетом android.widget[1]. В дальнейшем будут рассмотрены дополнительные компоненты GUI, которые могут использоваться в макетах. Чтобы получить полный перечень этих компонентов, обратитесь к веб-сайту developer.android.com/reference/android/widget/package-summary.html.

Таблица 3.3. Макеты Android (package android.widget)

Макет	Описание
FrameLayout	Выделяет пространство для одного компонента. В этот макет можно добавить несколько компонентов, причем каждый из них будет отображаться над правым верхним углом макета. Последний добавленный компонент отображается сверху
LinearLayout	Расположение компонентов по горизонтали в одной строке или по вертикали в одном столбце

продолжение ⇗

[1] В состав ранних версий Android SDK также входил макет AbsoluteLayout, в котором определялась точная позиция каждого компонента. Сейчас этот макет не рекомендуется к применению. Вместо этого макета рекомендуется использовать макет FrameLayout, RelativeLayout или пользовательский макет (см. developer.android.com/reference/android/widget/AbsoluteLayout.html).

Таблица 3.3 *(продолжение)*

Макет	Описание
RelativeLayout	Расположение компонентов относительно друг друга или относительно родительского контейнера
TableLayout	Расположение компонентов в виде таблицы, состоящей из строк. Для формирования столбцов таблицы можно воспользоваться макетом TableRow layout (подкласс макета LinearLayout)

ПРИМЕЧАНИЕ

Чтобы включить поддержку экранов, имеющих различные размеры и разрешения, для проектирования графического интерфейса пользователя используйте компоненты RelativeLayout и TableLayout.

Конфигурирование Visual Layout Editor для использования соответствующей библиотеки Android SDK

Если установлено несколько библиотек Android SDK, модуль ADT Plugin по умолчанию выбирает самую последнюю из них независимо от версии SDK, выбранной при создании проекта. Эта библиотека используется для проектирования графического интерфейса пользователя и находится на вкладке Graphical Layout (Графический макет). Обратите внимание на рис. 3.7, где выбрана библиотека Android 2.3.3 в раскрывающемся списке SDK selector (Выбор SDK), находящемся в правой верхней части вкладки Graphic Layout. В результате определяется разработка графического интерфейса пользователя для устройства Android 2.3.3.

Удаление и воссоздание файла main.xml

В процессе создания приложения для этой главы вместо заданного по умолчанию файла main.xml будет использоваться новый файл макета, RelativeLayout (в этом файле определяется относительное расположение компонентов). Чтобы заменить заданный по умолчанию файл main.xml, выполните следующие действия:

1. Убедитесь в том, что файл main.xml закрыт, потом щелкните на нем правой кнопкой мыши (в папке /res/layout проекта) и выберите команду Delete (Удалить) для удаления файла.
2. Чтобы отобразить диалоговое окно New... (Создать...), щелкните правой кнопкой мыши на папке макета и выберите команды New ▸ Other...(Создать ▸ Другой...).
3. В узле Android выберите параметр Android XML File (XML-файл Android) и щелкните на кнопке Next> (Далее>), чтобы отобразить диалоговое окно New Android XML File (Новый XML-файл Android).
4. Определите имя файла, его местоположение и корневой макет для нового файла main.xml (см. рис. 3.7), а потом щелкните на кнопке Finish (Готово).

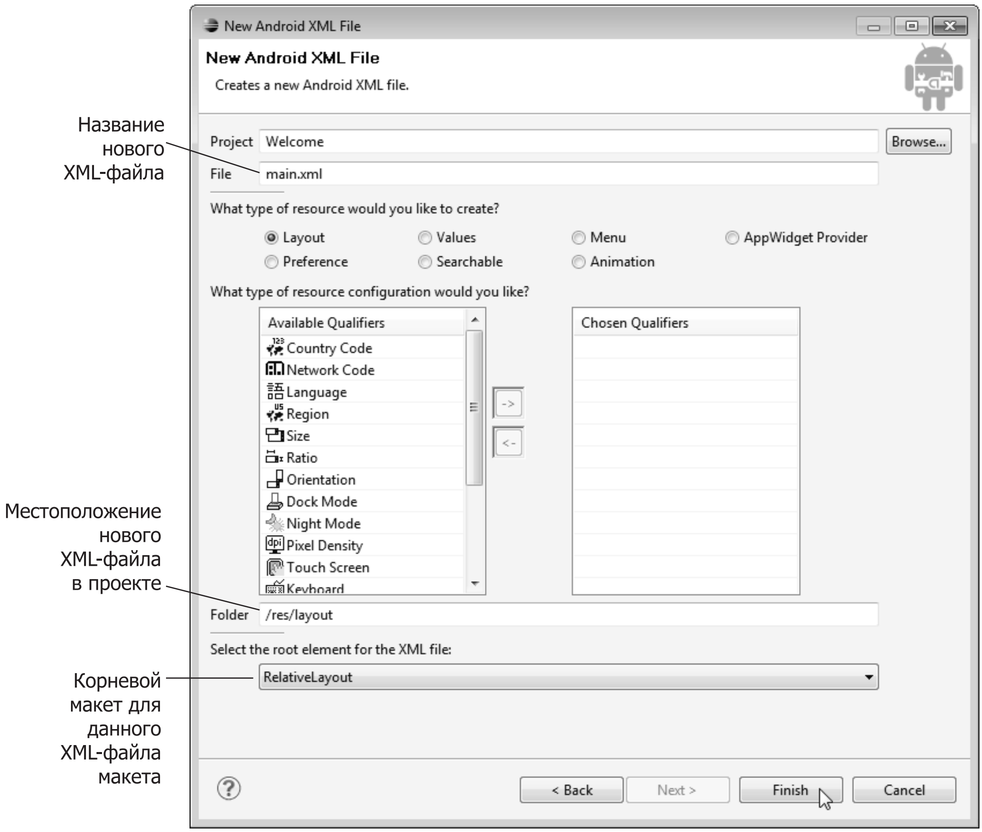


Рис. 3.7. Создание нового файла main.xml в диалоговом окне New Android XML File

Настройка размера и разрешения экрана для Visual Layout Editor

На рис. 3.8 показан новый файл `main.xml` в окне редактора Visual Layout Editor. Поскольку операционная система Android может выполняться на множестве различных устройств, в комплект поставки Visual Layout Editor входит ряд различных конфигураций устройств, представляющих разные размеры и разрешения экрана. Эти настройки выбираются в раскрывающемся списке `Device Configurations` (Конфигурации устройства). Этот раскрывающийся список находится в верхней левой части вкладки `Graphic Layout` (рис. 3.8). Если стандартные конфигурации не соответствуют устройству, для которого разрабатывается приложение, можно создать собственную конфигурацию устройства «с нуля» либо путем копирования и изменения одной из имеющихся конфигураций.

Для создания примеров в книге мы использовали основное тестовое устройство **Samsung Nexus S**, которое снабжено 4-дюймовым экраном с разрешением 480×800 пикселей (WVGA). Рекомендуется разрабатывать *масштабируемую* библиотеку Android GUI, которая позволит корректно формировать изображения на экранах различных устройств. Благодаря свойству масштабируемости не требуется точное соответствие

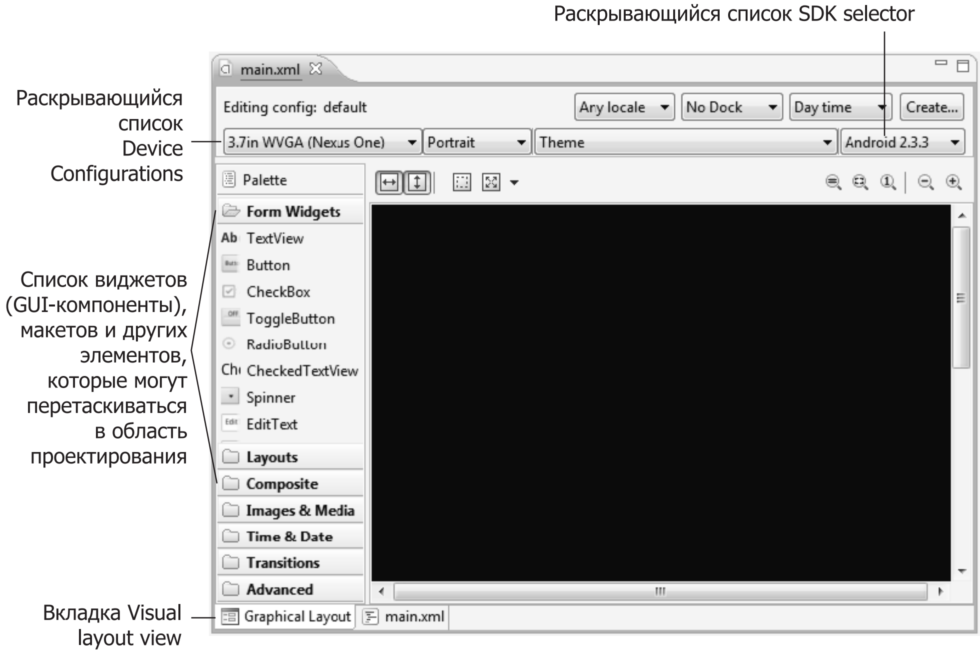


Рис. 3.8. Представление Visual Layout Editor для заданного по умолчанию GUI приложения

области проектирования Visual Layout Editor экранам физических устройств. Достаточно выбрать подобную конфигурацию устройства. На рис. 3.8 выбран параметр 3.7in WVGA (Nexus One). Экран этого устройства имеет то же разрешение WVGA, что и Nexus S, но немного меньший размер экрана. Экраны многих современных смартфонов имеют разрешение 480×800 или 480×854 пикселей.

Размеры и разрешение изображений и экрана

Поскольку экраны устройств Android имеют различные размеры, разрешения и пиксельные плотности экрана (выражается в точках на дюйм или DPI), поддерживаются изображения (и другие ресурсы), которые выбираются операционной системой в соответствии с пиксельной плотностью экрана устройства. Поэтому в папке res проекта находятся три подпапки — drawable-hdpi (высокая плотность), drawable-mdpi (средняя плотность) и drawable-ldpi (низкая плотность). В этих папках хранятся изображения с различными пиксельными плотностями (табл. 3.4).

Таблица 3.4. Пиксельные плотности устройств Android

Плотность	Описание
ldpi	Низкая плотность — примерно 120 точек на дюйм
mdpi	Средняя плотность — примерно 160 точек на дюйм

Плотность	Описание
hdpi	Высокая плотность — примерно 240 точек на дюйм
xhdpi	Экстравысокая плотность — примерно 320 точек на дюйм
nodpi	Отсутствие масштабирования ресурсов независимо от плотности экрана

Изображения для устройств, которые имеют ту же самую пиксельную плотность, что и тестовое устройство, находятся в папке drawable-hdpi. Изображения, предназначенные для экранов, имеющих среднюю и низкую пиксельную плотности, находятся в папках drawable-mdpi и drawable-ldpi соответственно. Как и в случае с Android 2.2, можно добавить подпапку drawable-xhdpi в папку приложения res, чтобы представить экраны, которые имеют экстравысокие пиксельные плотности. В Android может выполняться масштабирование изображений в целях приведения в соответствие с различными экранными плотностями.

ПРИМЕЧАНИЕ

Чтобы получить подробную информацию о поддержке нескольких экранов и размеров экранов на Android, обратитесь к веб-сайту developer.android.com/guide/practices/screens_support.html.

ПРИМЕЧАНИЕ

Чтобы улучшить отображение на экране устройства с высокой пиксельной плотностью, следует выбирать изображения с высоким разрешением. Изображения с низким разрешением могут некорректно масштабироваться.

Шаг 1. Добавление изображений в проект

А теперь приступим к созданию приложения Welcome. Для создания приложения будет использован редактор Visual Layout Editor и окно Outline (Макет), затем будет подробно рассмотрена структура сгенерированного XML-кода. В следующих главах мы приступим к непосредственному редактированию XML.

ПРИМЕЧАНИЕ

Многие профессиональные Android-разработчики предпочитают создавать GUI непосредственно в XML, а затем с помощью редактора Visual Layout Editor просматривать результаты его выполнения. В процессе ввода XML-кода в представлении XML Eclipse предлагает функции автозавершения, отображая имена компонентов, атрибутов и значения, которые соответствуют вводимым именам и значениям. Благодаря этому можно быстро и корректно вводить XML-код.

Для данного приложения в проект нужно добавить изображения Deitel bug (bug.png) и логотипа Android (android.png). Эти изображения находятся в папке images вместе с примерами книги. Чтобы добавить изображения в проект, выполните следующие действия:

1. В окне Package Explorer откройте папку проекта res.
2. Найдите и откройте папку images, которая находится среди примеров книги, потом перетащите изображения в папку, которая находится в подпапке drawable-hdpi папки res.

Теперь изображения могут использоваться в приложении.

Шаг 2. Изменения свойства Id макета RelativeLayout

С помощью окна Properties (Свойства) можно сконфигурировать свойства выбранного макета или компонента без непосредственного редактирования XML-кода. Если окно Properties не отображается, отобразите его двойным щелчком на пункте RelativeLayout в окне Outline (Структура). Можно также выполнить команды Window ▸ Show View ▸ Other... (Окно ▸ Показать представление ▸ Другое...), затем выбрать параметр Properties из узла General (Общий) в диалоговом окне Show View (Показать представление). Для выбора макета или компонента щелкните на нем в окне редактора Visual Layout Editor либо выберите его узел в окне Outline (рис. 3.9). Окно Properties не может быть использовано, если макет отображается в представлении XML.



Рис. 3.9. Иерархическое представление GUI в окне Outline

Переименуйте каждый макет и компонент. Это особенно полезно в том случае, если макет или компонент будут подвергаться программной обработке (см. следующие главы). Имя каждого объекта определяется с помощью его свойства Id. Это свойство может применяться для получения доступа и изменения компонента, при этом не нужно знать точное местоположение компонента в XML-коде. Как будет показано позднее, свойство Id также может применяться для указания относительного позиционирования компонентов в макете RelativeLayout.

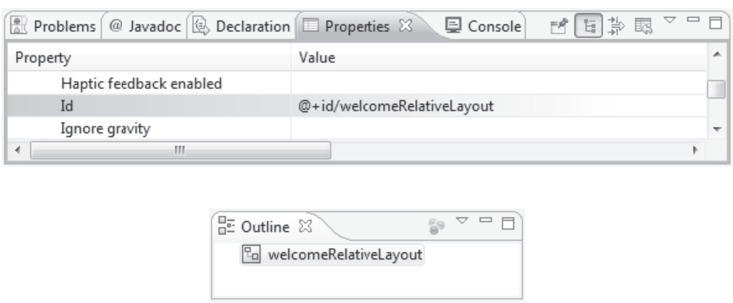


Рис. 3.10. Окно Properties, отображаемое после изменения свойства Id компонента RelativeLayout

Выберите компонент RelativeLayout, затем, выполнив прокрутку в окне Properties, выберите свойство Id и присвойте ему значение:

```
@+id/welcomeRelativeLayout
```

Символ + в конструкции @+id определяет создание нового идентификатора (имя переменной), который указан справа от символа /. На экране появятся окна Properties и Outline (рис. 3.10).

Шаг 3. Присваивание свойству Background значения RelativeLayout

По умолчанию в качестве фонового цвета макета выбирается черный, но его можно изменить (например, на белый). Любой цвет представляет собой результат смешения красного, синего и зеленого компонентов, которые называются **RGB-компонентами**. Каждый из этих компонентов может принимать целочисленное значение в диапазоне от 0 до 255. Первый компонент задает количество красного в общем цвете, второй — количество зеленого, а третий — количество синего цвета. Во время работы в интегрированной среде разработки (IDE) для определения цвета используется шестнадцатеричный формат. То есть RGB-компоненты представлены в виде значений в диапазоне 00–FF.

Чтобы изменить фоновый цвет, найдите свойство Background в окне Properties и присвойте ему значение #FFFFFF (рис. 3.11). Это значение представляет белый цвет в шестнадцатеричном формате. Формат #RRGGBB — это пары шестнадцатеричных чисел, которые представляют красный, зеленый и синий цвета соответственно. В Android также поддерживаются альфа-значения (прозрачность), которые изменяются в диапазоне 0–255. Значение 0 соответствует полной прозрачности, а значение 255 — полной непрозрачности. Если планируется использовать альфа-значения, укажите цвет в формате #AARRGGBB, где первые две шестнадцатеричных цифры соответствуют альфа-значению. Если обе цифры каждого компонента цвета одинаковы, можно воспользоваться форматом #RGB или #ARGB. Например, значение #FFF трактуется как #FFFFFF.

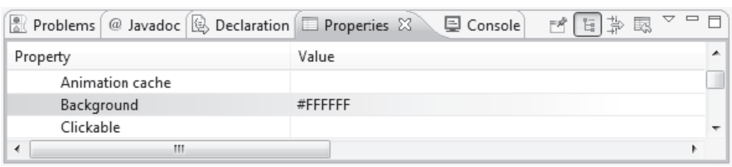


Рис. 3.11. Окно Properties после изменения значения свойства Background компонента RelativeLayout

Шаг 4. Добавление компонента TextView

На этом шаге добавим в пользовательский интерфейс компонент TextView. В списке Form Widgets (Виджеты форм), находящемся в левой части окна Visual Layout Editor, найдите компонент TextView и перетащите его в область проектирования (рис. 3.12). После добавления нового компонента в пользовательский интерфейс происходит его автоматический выбор и отображение свойств в окне Properties.

Компонент TextView с текстом, заданным по умолчанию

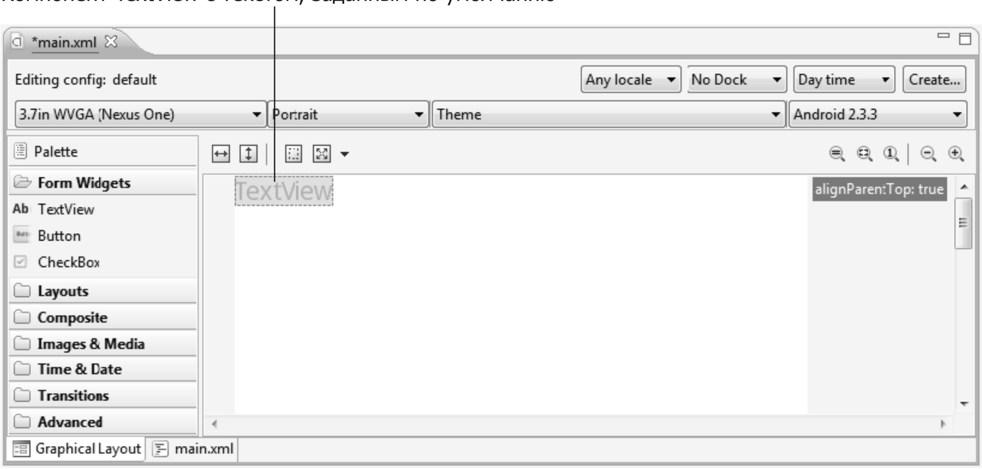


Рис. 3.12. Компонент TextView с текстом, заданным по умолчанию

Шаг 5. Настройка свойства Text компонента TextView с помощью строковых ресурсов

Согласно документации Android по ресурсам приложений (developer.android.com/guide/topics/resources/index.html) считается хорошим тоном хранить строки, массивы строк, изображения, цвета, размеры шрифтов, размерности и другие ресурсы приложения так, чтобы их можно было использовать отдельно от кода приложения. Например, после *экстернализации* цветовых значений все компоненты, использующие один и тот же цвет, могут быть перекрашены путем простого изменения значения цвета в центральном файле ресурса.

Если нужно локализовать приложения, создав версии на нескольких разных языках, сохраните строки отдельно от кода приложения. В результате у вас в дальнейшем появится возможность простого изменения этих строк. В папке res проекта находится подпапка values, в которой помещен файл strings.xml. Этот файл применяется для хранения строк. Чтобы сформировать локализованные строки для других языков, создайте отдельные папки values для каждого используемого языка.

Например, в папке values-fr может находиться файл strings.xml для французского языка, а в папке values-es — файл strings.xml для испанского языка. Можно также формировать названия этих папок с учетом информации о регионе. Например, в папке values-en-rUS может находиться файл strings.xml для американского диалекта английского языка, а в папке values-en-rGB — файл strings.xml для диалекта английского языка, используемого в Великобритании.

Чтобы получить дополнительные сведения о локализации, обратитесь к ресурсам:

- developer.android.com/guide/topics/resources/;
- providing-resources.html#AlternativeResources;
- developer.android.com/guide/topics/resources/localization.html.

Чтобы установить значение свойства Text компонента TextView, создадим новый строковый ресурс в файле strings.xml.

1. Выберите компонент TextView.
2. В окне Properties найдите свойство Text, щелкните на значении, заданном по умолчанию, затем щелкните на кнопке с многоточием. Эта кнопка находится в правой части поля значения свойства и служит для отображения диалогового окна Resource Chooser (Выбор ресурсов).
3. В диалоговом окне Resource Chooser щелкните на кнопке New String... (Новая строка...), чтобы отобразить диалоговое окно Create New Android String (Создать новую строку Android), показанное на рис. 3.13.
4. Заполните поля String и New R.string (см. рис. 3.13), потом щелкните на кнопке OK, чтобы скрыть диалоговое окно Create New Android String и вернуться к окну Resource Chooser.
5. Автоматически выбирается новый строковый ресурс welcome. Щелкните на кнопке OK для выбора этого ресурса.

После выполнения указанных выше шагов в окне Properties появляется свойство Text (рис. 3.14). Запись в форме @string свидетельствует о том, что существующий

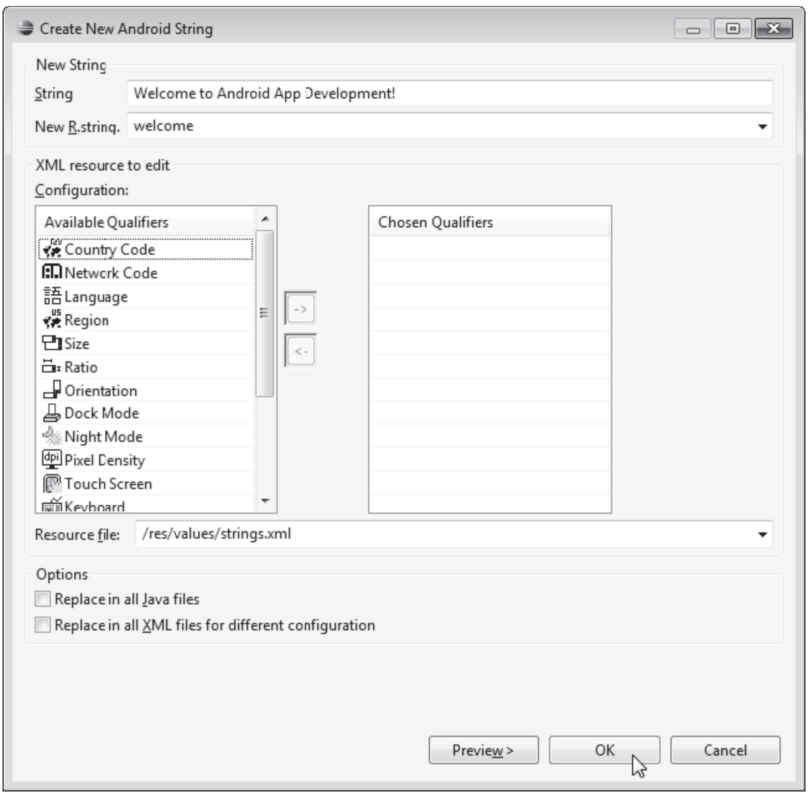


Рис. 3.13. Окно Create New Android String

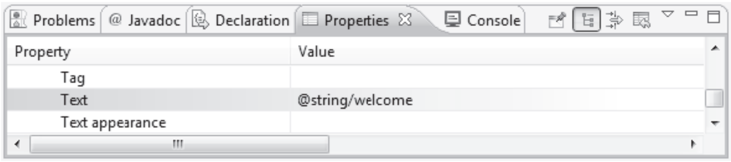


Рис. 3.14. Вид окна Properties после изменения свойства Text компонента TextView

строковый ресурс может быть выбран в файле strings.xml, а имя welcome говорит о том, что строковый ресурс выбран в данный момент времени.

Основное преимущество определения строковых значений заключается в облегчении *локализации* приложения, осуществляемой путем создания дополнительных файлов XML-ресурсов для других языков. В каждом файле используется одно и то же имя в поле New R.string и поддерживается интернационализированная строка в поле String. Потом Android может выбрать соответствующий файл ресурса, основываясь на языке, предпочтительном для пользователя устройства. Дополнительные сведения о локализации можно найти на веб-сайте developer.android.com/guide/topics/resources/localization.html.

Шаг 6. Настройка свойств Text size и Padding top компонента TextView — пиксели, независимые от плотности и от масштабирования

Размеры компонентов GUI и текста на экране Android могут определяться с помощью различных единиц измерения (табл. 3.5). Документация, описывающая различные размеры экранов, находится на веб-сайте по адресу developer.android.com/guide/practices/screens_support.html и рекомендует для определения размеров компонентов GUI и других экранных элементов использовать пиксели, независимые от плотности, а размеры шрифтов задавать с помощью пикселей, независимых от масштабирования.

Таблица 3.5. Единицы измерения

Единица измерения	Описание
px	Пиксель
dp или dip	Пиксель, независимый от плотности
sp	Пиксель, независимый от масштабирования
in	Дюймы
mm	Миллиметры

Задание размеров в пикселях, независимых от плотности (dp или dip), позволяет платформе Android автоматически масштабировать графический интерфейс пользователя в зависимости от плотности пикселей экрана физического устройства.

Размер пикселя, независимого от плотности, эквивалентен размеру физического пикселя на экране с разрешением 160 dpi (точек на дюйм). На экране с разрешением 240 dpi размер пикселя, независимого от плотности, будет масштабироваться на коэффициент 240/160 (то есть 1,5). Таким образом, компонент, размер которого составляет 100 пикселей, независимых от плотности, будет масштабирован до размера в 150 физических пикселей на таком экране. На экране с разрешением 120 точек на дюйм каждый независимый от плотности пиксель масштабируется на коэффициент 120/160 (то есть 0,75). Значит, 100 независимых от плотностей пикселей превратятся на таком экране в 75 физических пикселей. Пиксели, независимые от масштаба, масштабируются так же, как и пиксели, независимые от плотности, но их масштаб зависит также и от предпочитаемого размера шрифта, выбираемого пользователем.

ПРИМЕЧАНИЕ

На момент написания книги пользователи не могли изменять предпочитаемый размер шрифта, но эта ситуация изменится в ближайшем будущем.

А теперь увеличим размер шрифта для компонента TextView и добавим небольшой отступ над компонентом TextView, чтобы отделить текст от края экрана устройства.

1. Чтобы изменить размер шрифта, выберите компонент TextView, а потом присвойте свойству Text size значение 40sp.
2. Чтобы увеличить отступ между верхним краем макета и компонентом TextView, выберите свойство Layout margin top property в разделе Misc (Разное) окна Properties и присвойте ему значение 10dp.

Шаг 7. Настройка дополнительных свойств компонента TextView

Сконфигурируйте дополнительные свойства компонента TextView следующим образом:

1. Свойству Id присвойте значение @+id/welcomeTextView.
2. Свойству Text color присвойте значение #00F (синий).
3. Свойству Text style присвойте значение bold. Чтобы выполнить эту операцию, щелкните в области поля Value этого свойства, затем щелкните на кнопке с многоточием для отображения диалогового окна, в котором выбирается стиль шрифта. Установите флажок bold, потом щелкните на кнопке OK для выбора стиля текста.
4. Чтобы отцентрировать текст компонента TextView при его расположении в нескольких строках свойству Gravity присвойте значение center. Для выполнения этой операции щелкните в области поля Value этого свойства, потом щелкните на кнопке с многоточием для отображения диалогового окна параметров свойства Gravity (рис. 3.15). Установите флажок center и щелкните на кнопке OK для выбора значения.

   На экране снова появится окно Visual Layout Editor (рис. 3.16).

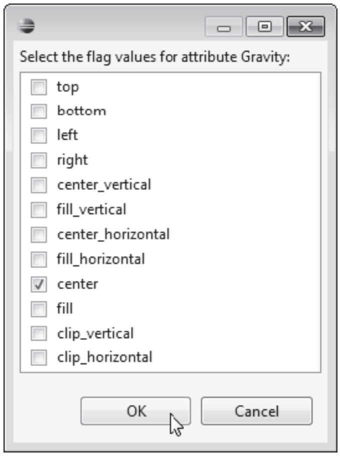


Рис. 3.15. Параметры атрибута gravity для объекта

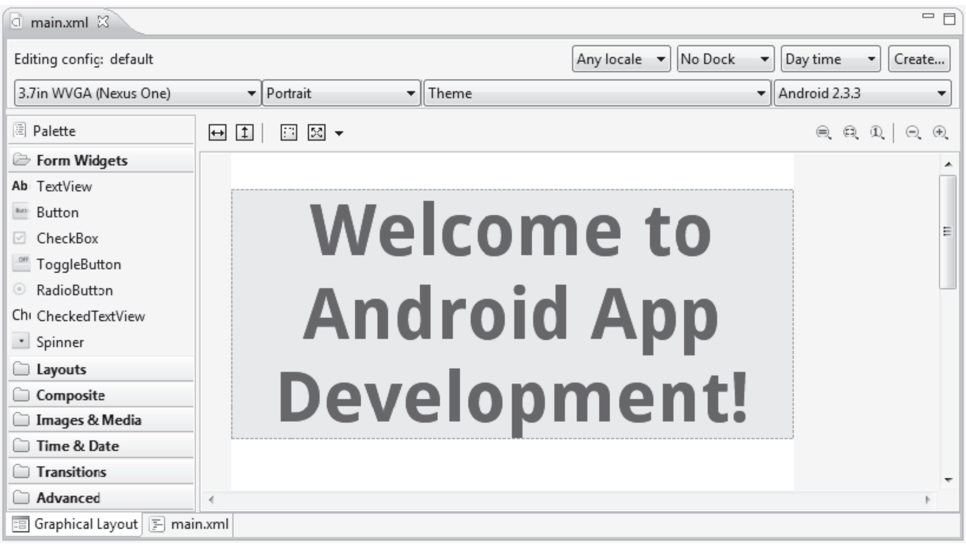


Рис. 3.16. Вид окна Visual Layout Editor после завершения конфигурирования компонента TextView

Шаг 8. Отображение логотипов Android и Deitel Bug с помощью компонентов ImageViews

А теперь в графический интерфейс пользователя добавим два компонента ImageViews, с помощью которых отображаются два изображения, добавленные в проект на шаге 1. После первого перетаскивания компонента ImageView в окно редактора Visual Layout

Editor ничего не происходит. Поэтому для добавления компонентов ImageViews используется окно Outline. Выполните следующие действия:

1. Перетащите компонент ImageView, находящийся в категории Images & Media (Изображения и медиаресурсы) палитры Visual Layout Editor, в окно Outline (Макет), как показано на рис. 3.17. Новый компонент ImageView появится ниже узла welcomeTextView. Вы не обнаружите какого-либо признака, свидетельствующего о том, что компонент появился ниже компонента TextView в GUI. Это означает, что требуется установить значение свойства Layout below, чем мы и займемся в скором времени.

ПРИМЕЧАНИЕ

Если перетащить компонент ImageView поверх элемента welcomeTextView и удерживать его в течение некоторого времени, вокруг элемента welcomeTextView появится зеленый прямоугольник с секциями. Если перетащить компонент ImageView поверх этих секций, редактор Visual Layout Editor может установить относительное позиционирование.



Рис. 3.17. Перетаскивание компонента ImageView в окно Outline

2. Свойству Id компонента ImageView присвойте значение @+id/droidImageView. В окне Outline отобразится название объекта droidImageView.
3. Свойству Layout below объекта droidImageView присвойте значение @id/welcomeTextView. В результате компонент ImageView будет расположен ниже объекта welcomeTextView. Чтобы выполнить эту операцию, щелкните в области поля Value

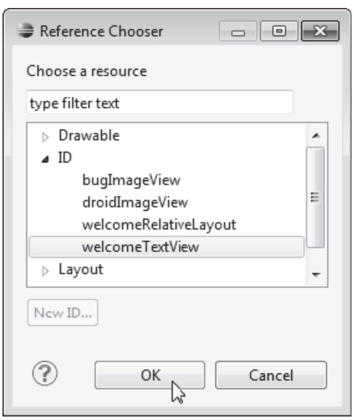


Рис. 3.18. Выбор значения для свойства Layout below объекта droidImageView

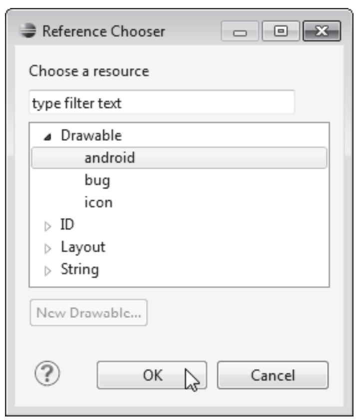


Рис. 3.19. Выбор значения для свойства Src объекта droidImageView

этого свойства, затем щелкните на кнопке с многоточием, чтобы отобразить диалоговое окно Reference Chooser (Выбор ссылок), показанное на рис. 3.18. В узле ID находятся названия объектов GUI. Раскройте узел ID и выберите объект welcomeTextView.

4. Присвойте свойству Layout center horizontal объекта droidImageView значение true, чтобы отцентрировать компонент ImageView внутри макета.
5. Присвойте свойству Src объекта droidImageView изображение, которое должно быть отображено. Для выполнения этой операции щелкните в области поля Value данного

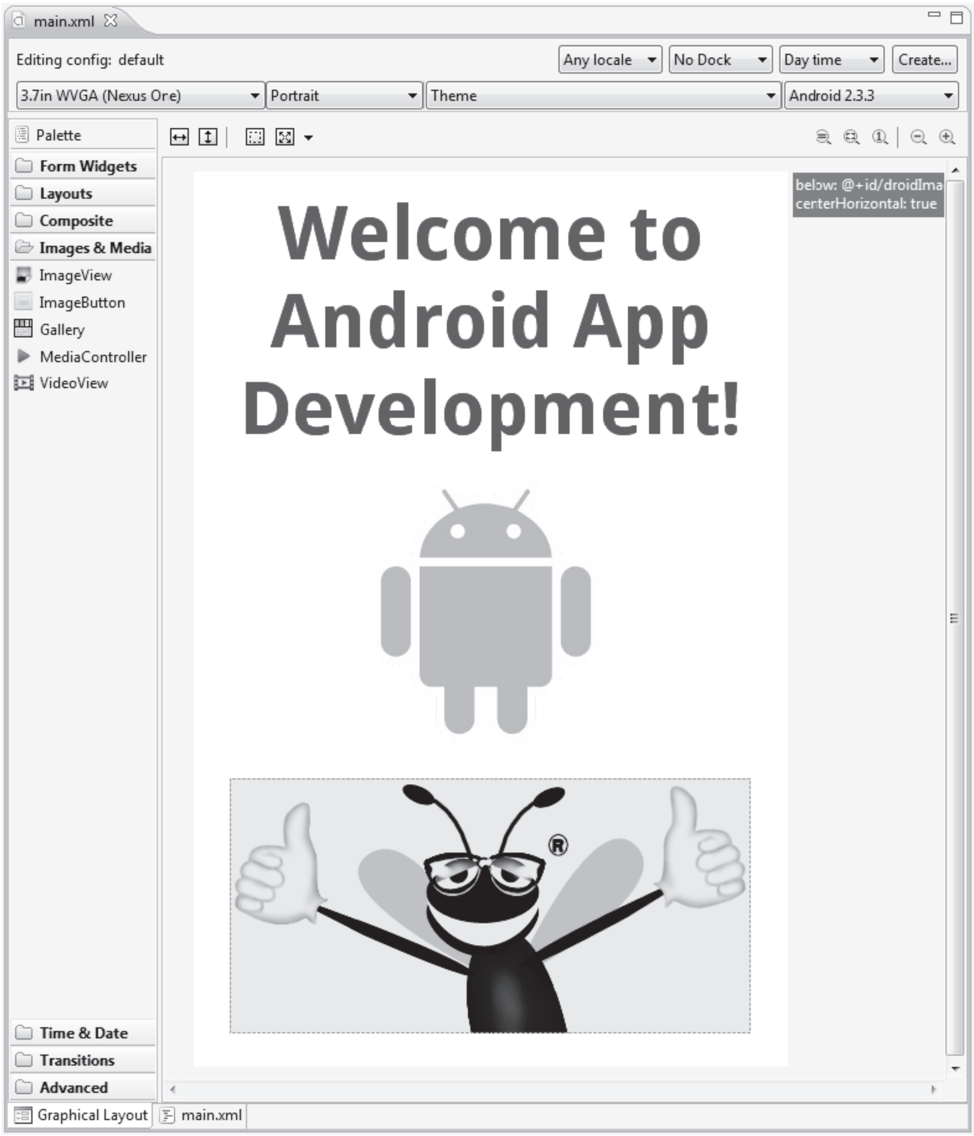


Рис. 3.20. Окно Visual Layout Editor после завершения конфигурирования GUI

свойства, а потом щелкните на кнопке с многоточием, чтобы отобразить диалоговое окно Reference Chooser (рис. 3.19). В узле Drawable расположены ресурсы, находящиеся в папках приложения drawable, которые находятся в папке res. В диалоговом окне раскройте узел Drawable и выберите параметр android, который представляет изображение android.png.

6. Повторите предыдущие пункты 1–5 для создания компонента bugImageView. Присвойте свойству Id компонента значение @+id/bugImageView, свойству Src — значение bug, а свойству Layout below — значение droidImageView.

Теперь окно Visual Layout Editor будет выглядеть так, как показано на рис. 3.20.

3.6. Структура файла main.xml

Язык XML позволяет естественным образом описать контент GUI. В формате, воспринимаемом компьютером и человеком, этот язык «говорит» о том, какие макеты и компоненты вы желаете использовать, а также позволяет определить их атрибуты, такие как размер, положение и цвет. Потом подключаемый модуль ADT Plugin анализирует XML и генерирует код, который реализует графический интерфейс пользователя. В листинге 3.1 приведен финальный код файла main.xml, сгенерированный после выполнения шагов, описанных в разделе 3.5. Мы выполнили форматирование XML-кода и добавили комментарии, чтобы повысить удобство чтения XML-кода. (Чтобы выполнить форматирование, воспользуйтесь командой Eclipse Source ▸ Format (Исходный код ▸ Формат).) В процессе чтения XML-кода нетрудно заметить, что имя каждого атрибута XML, состоящее из нескольких слов, не включает пробелы, в то время как названия соответствующих свойств в окне Properties могут включать пробелы. Например, атрибут XML android:paddingTop соответствует свойству Padding top в окне Properties. В среде IDE имена свойств, состоящие из нескольких слов, отображаются раздельно с целью улучшения читабельности.

Листинг 3.1. XML-разметка приложения Welcome

```
<?xml version="1.0" encoding="utf-8"?>
<!-- main.xml -->
<! — XML-разметка приложения Welcome. -->

<! — RelativeLayout, включающий компоненты GUI приложения -->
<RelativeLayout xmlns:android=http://schemas.android.com/apk/res/android
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:id="@+id/welcomeRelativeLayout" android:background="#FFFFFF">

 <! — TextView, отображающий "Welcome to Android App Development!" -->
 <TextView android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/welcome"
    android:textSize="40sp" android:id="@+id/welcomeTextView"
    android:textColor="#00F" android:textStyle="bold"
    android:layout_centerHorizontal="true" android:gravity="center"
    android:layout_marginTop="10dp"></TextView>
```

продолжение ⇗

Листинг 3.1 *(продолжение)*

```
19
20  <! — ImageView, который отображает логотип Android -->
21  <ImageView android:layout_height="wrap_content"
22     android:layout_width="wrap_content" android:id="@+id/droidImageView"
23     android:layout_centerHorizontal="true"
24     android:src="@drawable/android"
25     android:layout_below="@id/welcomeTextView"></ImageView>
26
27  <! — ImageView, отображающий логотип Deitel bug -->
28  <ImageView android:layout_height="wrap_content"
29     android:layout_width="wrap_content" android:id="@+id/bugImageView"
30     android:src="@drawable/bug"
31     android:layout_below="@id/droidImageView"
32     android:layout_centerHorizontal="true"></ImageView>
33 </RelativeLayout>
```

welcomeRelativeLayout

Блок welcomeRelativeLayout (строки 6–33) содержит все компоненты GUI приложения.

- Открывающий тег XML (строки 6–9) устанавливает различные атрибуты RelativeLayout.
- В строке 6 используется атрибут xmlns, который указывает все элементы в документе, относящиеся к пространству имен android XML. Этот атрибут обязателен для указания и автоматически генерируется IDE при создании любого файла XML-разметки.
- В строках 7–8 указывается значение match_parent для атрибутов android:layout_width и android:layout_height. В результате макет занимает всю высоту и ширину родительского элемента макета (элемента, в который вложен данный макет). В данном случае RelativeLayout является корневым узлом документа XML, в результате чего макет занимает все пространство экрана (помимо строки состояния).
- В строке 9 указываются значения атрибутов android:id и android:background для компонента welcomeRelativeLayout.

welcomeTextView

Первый элемент компонента welcomeRelativeLayout — это welcomeTextView (строки 12–18).

- В строках 12 и 13 атрибутам android:layout_width и android:layout_height присваивается значение wrap_content. Это значение свидетельствует о том, что представление будет достаточно большим, чтобы вместить все содержимое, включая значения отступа, с помощью которых определяется отступ вокруг содержимого.

- В строке 14 устанавливается атрибут android:text таким образом, чтобы ему соответствовал строковый ресурс welcome, который был создан на шаге 5 раздела 3.5.
- В строке 15 атрибуту android:textSize присваивается значение 40sp, а атрибуту android:id — значение "@+id/welcomeTextView".
- В строке 16 атрибуту android:textColor присваивается значение "#00F" (выделенный текст), а атрибуту android:textStyle — значение "bold".
- В строке 17 атрибуту android:layout_centerHorizontal присваивается значение "true", которое центрирует компонент по горизонтали в макете. Атрибуту android:gravity присваивается значение "center", которое центрирует текст компонента TextView. Атрибут android:gravity определяет, каким образом позиционируется текст относительно ширины и высоты компонента TextView, если текст будет меньше, чем TextView.
- В строке 18 атрибуту android:marginTop присваивается значение 10dp, в результате чего добавляется дополнительное пространство между верхней частью компонента TextView и экрана.

droidImageView

Последние два элемента, вложенные в компонент welcomeRelativeLayout, — это droidImageView (строки 21–25) и bugImageView (строки 28–32). Для двух компонентов ImageViews используются одни и те же атрибуты, поэтому будут рассмотрены лишь атрибуты компонента droidImageView.

- В строках 21 и 22 атрибутам android:layout_width и android:layout_height присваивается значение wrap_content. В строке 22 атрибуту android:id присваивается значение "@+id/droidImageView".
- В строке 23 атрибуту android:layout_centerHorizontal присваивается значение "true", чтобы центрировать компонент в области макета.
- В строке 24 атрибуту android:src присваивается ресурс, находящийся в папке drawable и имеющий название android. Этот атрибут представляет изображение android.png.
- В строке 25 атрибуту android:layout_below присваивается значение "@id/welcomeTextView".

Компонент RelativeLayout определяет положение каждого компонента относительно других компонентов. В этом случае компонент ImageView следует за компонентом welcomeTextView.

3.7. Выполнение приложения Welcome

Чтобы выполнить приложение на виртуальном устройстве Android Virtual Device (AVD), щелкните правой кнопкой мыши на корневом узле приложения в окне Package Explorer и выполните команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android). На рис. 3.21 показано запущенное приложение.

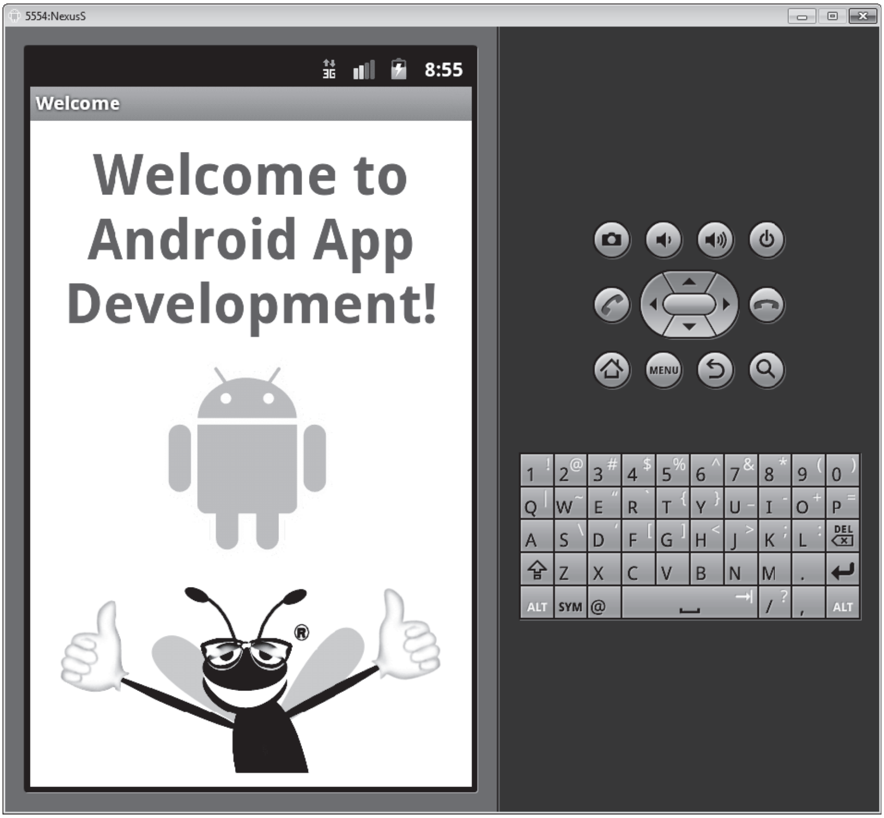


Рис. 3.21. Приложение Welcome, выполняющееся на виртуальном устройстве AVD

3.8. Резюме

В этой главе были рассмотрены ключевые функции интегрированной среды разработки Eclipse IDE и подключаемого модуля ADT для редактора Visual Layout Editor. С помощью Visual Layout Editor было создано работоспособное приложение Android без написания какого-либо кода. Мы воспользовались компонентами GUI `TextView` и `ImageView` для отображения текста и изображений соответственно. Эти компоненты были упорядочены в макете `RelativeLayout`. Мы отредактировали свойства GUI-компонентов, чтобы настроить их для приложения. Затем созданное приложение было протестировано на виртуальном устройстве Android Virtual Device (AVD). И наконец, был выполнен подробный анализ XML-разметки, которая генерирует GUI.

В следующей главе мы начнем разрабатывать Android-приложения с помощью Java. Разработка приложений для Android представляет собой сочетание проектирования графического интерфейса пользователя и кодирования на Java и XML. С помощью Java можно определить «поведение» ваших приложений.

Мы разработаем приложение Tip Calculator для подсчета размера чаевых для различных ресторанов. Мы разработаем графический интерфейс пользователя (GUI) и создадим код Java, с помощью которого приложение будет обрабатывать введенные пользователем данные и отображать результаты этих вычислений.

4 Приложение Tip Calculator App

Создание приложения Android с помощью Java

В этой главе...

- Создание графического интерфейса пользователя с помощью макета TableLayout.
- Использование окна Outline модуля ADT Plugin в среде Eclipse для добавления компонентов GUI в макет TableLayout.
- Непосредственное редактирование кода XML макета GUI для настройки свойств, которые могут быть недоступны в Visual Layout Editor и в окне Properties для Eclipse.
- Использование компонентов GUI TextView, EditText и SeekBar.
- Применение свойств объектно-ориентированного программирования на Java, включая классы, анонимные внутренние классы, объекты, интерфейсы и наследование, применяемые для создания приложения Android.
- Изменение отображаемого текста путем программного взаимодействия с компонентами GUI.
- Использование обработки событий при взаимодействии с пользователем с помощью компонентов EditText и SeekBar.

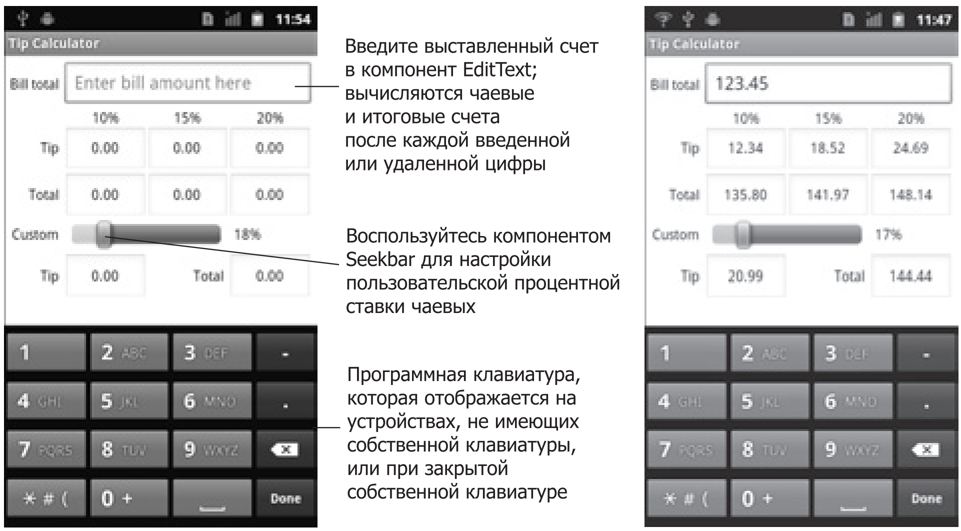
4.1. Введение

Приложение Tip Calculator (рис. 4.1) вычисляет и отображает чаевые на основании суммы счета в ресторане. После ввода пользователем суммы счета приложение вычисляет и отображает размер чаевых и величину итогового счета (с учетом чаевых). При этом используются три процентных ставки для расчета чаевых: 10 %, 15 % и 20 %. Пользователь может указать собственную ставку, используемую при расчете чаевых. Для этого

нужно переместить ползунок компонента Seekbar, после чего обновляется величина процентной ставки, которая отображается в правой части компонента Seekbar. Мы выбрали величину 18 % в качестве заданной по умолчанию пользовательской процентной ставки для чаевых. Это связано с тем, что многие рестораны используют эту процентную ставку для компаний, состоящих из шести человек и более. Предполагаемые величины чаевых и общего счета могут быть изменены пользователем.

ПРИМЕЧАНИЕ

> Цифровая клавиатура, показанная на рис. 4.1, может отличаться от вашей в зависимости от используемого устройства AVD либо версии Android, установленной на устройстве.



а *б*

Рис. 4.1. Ввод суммы счета и подсчет чаевых: *а* — начальный интерфейс пользователя, появляющийся после касания пользователем компонента EditText Bill total и отображения цифровой клавиатуры; *б* — графический интерфейс после ввода выставленного счета на $123,45 и изменения процентной ставки чаевых на 17 %

Начнем с тестирования приложения, заключающееся в вычислении стандартной и заданной пользователем ставок чаевых. Затем вас ожидает обзор технологий, применяемых для создания приложения. После этого мы создадим графический интерфейс пользователя (GUI) с помощью окна Outline в Eclipse. Это окно будет использоваться для добавления компонентов GUI. В графическом редакторе Visual Layout Editor мы увидим, как выглядит будущий GUI. Большая часть XML-кода для этого GUI будет сгенерирована с помощью инструментов плагина ADT. Потом мы отредактируем XML-код, что позволит настроить свойства, недоступные в окне Properties. И напоследок рассмотрим окончательный код приложения и проведем его подробный анализ.

4.2. Тестирование приложения Tip Calculator

Откройте и выполните приложение

Откройте интегрированную среду разработки Eclipse, импортируйте проект Tip Calculator и выполните следующие действия:

1. *Откройте диалоговое окно* Import (*Импорт*). Чтобы открыть диалоговое окно Import, выполните команды File ▸ Import... (Файл ▸ Импорт...).
2. *Импортируйте проект приложения* Tip Calculator. В диалоговом окне Import раскройте узел General (Общие), выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область), а потом щелкните на кнопке Next> (Далее>) для выполнения шага Import Projects (Импорт проектов). Установите флажок Select root directory (Выбрать корневой каталог) и щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папок) найдите папку TipCalculator (в папке примеров книги), выделите ее и щелкните на кнопке OK. Щелкните на кнопке Finish (Готово) для выполнения импорта проекта в среду Eclipse. Проект появится в окне Package Explorer, находящемся в левой части окна Eclipse.
3. *Запустите приложение* Tip Calculator. В окне среды Eclipse щелкните правой кнопкой мыши на проекте TipCalculator в окне Package Explorer, затем в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android). В результате приложение Tip Calculator начнет выполняться на виртуальном устройстве AVD, которое было создано так, как описано в разделе «Подготовительные действия» из вводной части книги.

ПРИМЕЧАНИЕ

Если было создано несколько устройств AVD либо к компьютеру подключено какое-либо физическое устройство Android, выберите одно из них для выполнения приложения.

Ввод выставленного счета

Коснитесь компонента EditText, который называется Bill Total, чтобы отобразить числовую клавиатуру. С помощью этой клавиатуры введите число 123,45.

ПРИМЕЧАНИЕ

Если нажатие клавиш приводит к появлению японских иероглифов, коснитесь пальцем элемента EditText под названием Bill Total и удерживайте палец пару секунд. В появившемся списке параметров выберите параметр Input method (Метод ввода). Потом выберите клавиатуру Android из второго списка параметров.

Если при вводе допущена ошибка, нажмите клавишу удаления, чтобы удалить последнюю введенную цифру. Компоненты EditText , имеющие заголовки 10 %, 15 % и 20 %, отображают размеры чаевых и общий счет для заранее заданных процентов чаевых (рис. 4.1, *б*). Также на экране отображаются компоненты EditText для пользовательских чаевых, общей суммы чаевых и общего счета для заданного по умолчанию значения 18 %, используемого при вычислении пользовательских чаевых. Все компоненты

EditText, используемые для вычисления чаевых и счета, обновляются после каждого ввода или удаления числа.

Выбор процентной ставки для пользовательских чаевых

Воспользуйтесь компонентом Seekbar для ввода процентной ставки, используемой при вычислении пользовательских чаевых. Перетаскивайте ползунок Seekbar до тех пор, пока процентная ставка для пользовательских чаевых не будет равна 17 %. Сумма чаевых и общего счета для выбранной процентной ставки появится в компонентах EditText, находящихся ниже компонента Seekbar. По умолчанию компонент Seekbar позволяет выбирать значения, находящиеся в диапазоне от 0 до 100.

4.3. Обзор применяемых технологий

В этой главе будут использоваться многие объектно-ориентированные свойства Java, включая классы, анонимные внутренние классы, объекты, методы, интерфейсы и наследование. Мы создадим подкласс класса Android Activity, с помощью которого определяется, что произойдет в случае запуска приложения на выполнение, а также логика приложения Tip Calculator. Будет реализовано программное взаимодействие с компонентами EditText, TextView и SeekBar. Эти компоненты создаются с помощью Visual Layout Editor и окна Outline в Eclipse, а также ручного редактирования XML-разметки, используемой для создания GUI. Компонент EditText, который часто называется текстовым полем или текстовым окном в других GUI-технологиях, — это подкласс компонента TextView (представлен в главе 3). Компонент EditText отображает текст либо принимает текст, вводимый пользователем. Компонент SeekBar, в других GUI-технологиях называемый ползунком, представляет целое число, заданное по умолчанию в диапазоне 0–100, которое может быть выбрано пользователем. Для организации взаимодействия между пользователем и интерфейсом пользователя применяется обработка событий и анонимные внутренние классы.

4.4. Создание графического интерфейса приложения

В этом разделе создается графический интерфейс приложения Tip Calculator с помощью инструментов модуля ADT Plugin. В конце этого раздела представлен код XML, сгенерированный модулем ADT Plugin для макета этого приложения. Мы подробно рассмотрим шаги, используемые для создания графического интерфейса. В следующих главах книги будут описаны новые свойства графического интерфейса каждого приложения, а также представлена завершенная XML-разметка интерфейса приложения с выделенными измененными фрагментами.

ПРИМЕЧАНИЕ

> После выполнения операций, описанных в этом разделе, будет сформирован графический интерфейс, который будет отличаться от интерфейса на рис. 4.1 до тех пор, пока не будут выполнены шаги, описанные в разделах 4.4.2–4.4.4.

4.4.1. Знакомство с классом TableLayout

В приложении, создаваемом в этой главе, используется объект TableLayout (см. рис. 4.2) для расположения компонентов GUI в шести строках и четырех столбцах. Каждая ячейка, определенная объектом TableLayout, может быть пустой или включать один компонент, в качестве которого может выступать макет, *содержащий* другие компоненты. В строках 0 и 4 на рис. 4.2 показан компонент, который может *охватывать* несколько столбцов. Для создания строк используются объекты TableRow. Количество столбцов в TableLayout определяется с помощью объекта TableRow, который содержит большинство компонентов. Высота каждой строки определяется наиболее длинным компонентом в этой строке (обратите внимание на рис. 4.2, где строки 1 и 4 короче других строк). Аналогично ширина столбца определяется наиболее широким элементом этого столбца (если не разрешить «растягивание» столбцов таблицы для заполнения экрана по ширине, в результате чего столбцы становятся шире). По умолчанию компоненты добавляются в строку в направлении слева направо. Можно указать точное положение компонента с помощью нумерации, которая по умолчанию начинается с 0.

Дополнительные сведения о классе TableLayout можно найти на веб-сайте: developer.android.com/reference/android/widget/TableLayout.html, а сведения о классе TableRow — на веб-сайте: developer.android.com/reference/android/widget/TableRow.html.

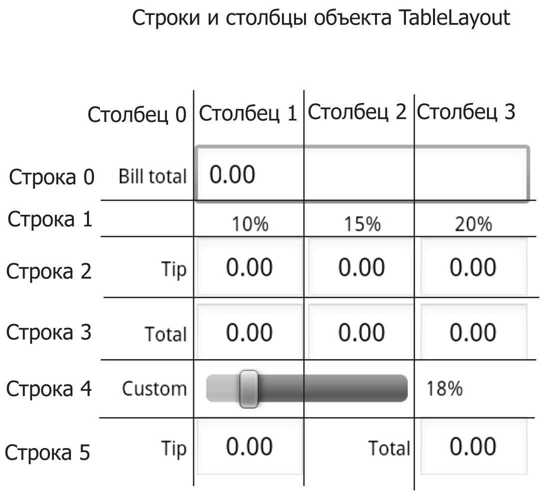


Рис. 4.2. Класс TableLayout с подписанными строками и столбцами, используемый для создания GUI приложения Tip Calculator

На рис. 4.3 показаны имена всех компонентов GUI, применяемых для формирования графического интерфейса приложения. Используемое соглашение о наименовании предусматривает использование имени класса компонента GUI в качестве значения свойства Id каждого компонента в XML-разметке и в имени переменной каждого компонента в коде Java.

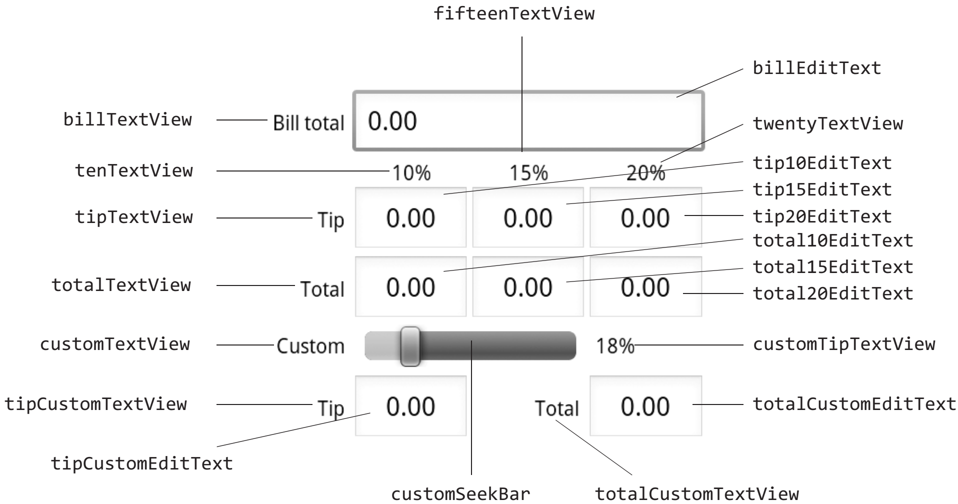


Рис. 4.3. Компоненты GUI приложения Tip Calculator, в качестве названий которых используются значения свойства Id

4.4.2. Создание проекта, добавление класса TableLayout и компонентов

А теперь приступим к созданию GUI, показанного на рис. 4.2. Начнем с формирования базового макета и элементов управления, потом настроим свойства элементов управления, завершив тем самым проект. После добавления компонентов в каждую строку объекта TableLayout можно настроить свойства Id и Text этих компонентов (см. рис. 4.3). В разделе 3.5 предыдущей главы упоминалось о том, что при разработке проекта значения литеральных строк должны включаться в файл strings.xml, который находится в папке res/values приложения. Эта методика особенно полезна в тех случаях, когда вы намереваетесь локализовать приложение для его использования на разных языках. Для компонентов TextView с названиями 10 %, 15 % и 20 % строковые ресурсы не используются. Убедитесь в том, что выполнили шаги, требуемые для создания GUI, компоненты которого расположены в заданном порядке. Если заданный порядок нарушен, измените расположение компонентов в окне Outline либо в файле main.xml.

При выполнении следующих шагов с помощью окна Outline выполняется добавление компонентов в соответствующие объекты TableRow из класса TableLayout. При работе с более сложными структурами, например TableLayouts, с помощью Visual Layout Editor довольно трудно просматривать сложную многоуровневую структуру макета и помещать компоненты на нужный уровень. В окне Outline отображается разветвленная структура GUI, в связи с чем задача проектирования существенно упрощается. Например, можно выбрать соответствующую строку в TableLayout и добавить в эту строку выбранный вами компонент GUI.

Шаг 1. Создание проекта TipCalculator

В среде Eclipse допускается создание лишь одного проекта с заданным именем для одной рабочей области. Поэтому перед выполнением этого шага удалите из рабочей области существующее приложение Tip Calculator, которое ранее уже тестировалось. Для выполнения этого действия щелкните на названии приложения правой кнопкой мыши и в контекстном меню выберите параметр Delete (Удалить). В появившемся диалоговом окне отмените установку флажка Delete project contents on disk is not selected (Удалить содержимое проекта на диске) и щелкните на кнопке OK. В результате произойдет удаление проекта из рабочей области, а папка проекта, находящаяся на диске, останется нетронутой. Затем создайте новый проект Android под названием TipCalculator. В диалоговом окне New Android Project (Новый проект Android) укажите следующие значения и щелкните на кнопке Finish (Готово):

- Build Target (Операционная система): выберите Android 2.3.3.
- Application name (Имя приложения): Tip Calculator.
- Package name (Название пакета): com.deitel.tipcalculator.
- Create Activity (Создать деятельность): TipCalculator.
- Min SDK Version (Минимальная версия SDK): 10. (*Примечание*. Эта версия SDK соответствует версии Android 2.3.3, хотя в этом приложении не будут использоваться функции, присущие именно Android 2.3.3. Если планируется выполнять это приложение на виртуальном устройстве Android (AVD) либо на реальном устройстве, на котором выполняется более ранняя версия Android, параметру Min SDK присвойте меньшее значение. Например, если выбрано значение 8 для этого параметра, приложение будет выполняться на версии Android 2.2 или более старшей.)

Шаг 2. Удаление и повторное создание файла main.xml

Для создаваемого в этой главе приложения нужно заменить стандартный файл main.xml другим, использующим TableLayout, в котором компоненты расположены друг относительно друга. Чтобы заменить стандартный файл main.xml, выполните следующие действия:

1. Чтобы удалить файл mail.xml, щелкните на нем (в папке проектов /res/layout) и в контекстном меню выберите параметр Delete (Удалить).
2. Правой кнопкой мыши щелкните на папке layout и выберите команды New ▸ Other... (Создать ▸ Другое...). Отобразится диалоговое окно New (Создать).
3. В узле Android выберите параметр Android XML File (XML-файл Android). После щелчка на кнопке Next> (Далее>) появится диалоговое окно New Android XML File (Создать XML-файл Android).
4. Выберите название файла main.xml и выделите TableLayout, потом щелкните на кнопке Finish.

Шаг 3. Конфигурирование Visual Layout Editor для использования подходящей библиотеки Android SDK

После выполнения предыдущего шага в окне Visual Layout Editor открывается новый файл main.xml. Если установлено несколько библиотек Android SDK, модуль ADT

Plugin выбирает последнюю библиотеку в качестве стандартной, используемой для создания интерфейса (на вкладке Graphical Layout), причем этот выбор независим от SDK, выбранной при создании проекта. В раскрывающемся списке селектора SDK, находящемся в правой верхней части вкладки Graphical Layout, выберите параметр Android 2.3.3 (см. рис. 3.7). В результате разрабатываемый графический интерфейс будет совместим с устройствами Android 2.3.3.

Шаг 4. Настройка размеров и разрешения окна Visual Layout Editor

В раскрывающемся списке Device Configurations (Конфигурации устройств), находящемся в левой верхней части вкладки Graphical Layout (см. рис. 3.11), выберите параметр 3.7in WVGA (Nexus One). В результате будет настроена область конфигурирования для устройств с экраном, имеющим разрешение 480×800 (WVGA).

Шаг 5. Конфигурирование параметров объекта TableLayout

В окне Outline выберите параметр TableLayout. На экране появится окно Properties, в котором представлены следующие свойства.

- Background (Фон): #FFF.
- Id (Идентификатор): @+id/tableLayou.t.
- Padding (Отступ): 5 dp.
- Stretch columns (Растяжение столбцов): 1, 2, 3.

По умолчанию параметрам Layout width и Layout height присваивается значение match_parent, в результате чего макет занимает весь экран. Параметру Padding присваивается значение 5 dp, в результате чего вдоль границы всего макета создается кайма шириной в 5 независимых от плотности пикселей. С помощью параметра Stretch columns (представлен в XML атрибутом android:stretchColumns, см. листинг 4.1, строка 8) определяется растяжение по горизонтали столбцов 1–3, чтобы заполнить макет по ширине. Ширина столбца 0 соответствует ширине наиболее широкого содержащегося в нем элемента вместе с отступом вокруг этого элемента.

Шаг 6. Добавление компонентов TableRow

На этом шаге с помощью окна Outline добавим шесть компонентов TableRow в объект TableLayout. Выполните следующие действия:

1. В окне Outline щелкните правой кнопкой мыши на компоненте tableLayout и выберите добавляемый компонент Outline.
2. Повторите этот процесс еще пять раз.

При каждом повторении процесса не забудьте щелкнуть правой кнопкой мыши на tableLayout с тем, чтобы TableRow корректно вкладывались в TableLayout. Свойству Id для каждого TableRow автоматически присваиваются значения от tableRow1 до tableRow6 соответственно. Поскольку нумерация столбцов начинается с 0, соответствующим образом были изменены свойства Id для TableRow, которые получили значения от tableRow0 до tableRow5 соответственно. Также выберите каждый объект TableRow и присвойте его свойству Layout width значение match_parent. В результате строки займут все пространство макета. Чтобы выполнить эту операцию для всех компонентов TableRow

одновременно, щелкните на первом компоненте TableRow, отображенном в окне Outline. Затем, удерживая клавишу Shift, щелкните на последнем компоненте TableRow в окне Outline, чтобы выделить все шесть компонентов. Затем присвойте значения свойствам.

Шаг 7. Добавление компонентов в tableRow

На этом шаге компоненты TextView и EditText будут добавлены в tableRow0. Выполните следующие действия:

1. Перетащите компонент TextView (billTextView) из раздела Form Widgets (Виджеты форм) палитры в tableRow0, находящийся в окне Outline.
2. Перетащите компонент EditText (billEditText) из раздела Form Widgets (Виджеты форм) палитры в tableRow0, находящийся в окне Outline.
3. Присвойте значения свойствам Id и Text для каждого компонента. Чтобы получить быстрый доступ к этим свойствам, щелкните правой кнопкой мыши на компоненте в окне Outline и в контекстном меню выберите команды Edit ID...(Изменить идентификатор...) и Edit Text...(Изменить текст...) соответственно.

Важно перетащить указанные выше компоненты на подходящие объекты TableRow в окне Outline, чтобы они были вложены в требуемый объект TableRow.

Шаг 8. Добавление компонентов в tableRow1

Добавьте три компонента TextView в объект tableRow1, выполнив следующие действия:

1. Перетащите компонент TextView (tenTextView) на объект tableRow1, находящийся в окне Outline.
2. Повторите этот процесс, добавив компоненты fifteenTextView и twentyTextView.
3. Установите значения свойств Id и Text для каждого компонента.

Шаг 9. Добавление компонентов в tableRow2

Добавьте компонент TextView и три компонента EditText в tableRow2, выполнив следующие действия:

1. В окне Outline перетащите компонент TextView (tipTextView) на tableRow2.
2. В окне Outline перетащите три компонента EditText на tableRow2 (tip10EditText, tip15EditText и tip20EditText).
3. Установите значения свойств Id и Text для каждого компонента.

Шаг 10. Добавление компонентов в tableRow3

Добавьте компонент TextView и три компонента EditText в tableRow3, выполнив следующие действия:

1. В окне Outline перетащите компонент TextView (totalTextView) на tableRow3.
2. В окне Outline перетащите три компонента EditText на tableRow3 (total10EditText, total15EditText и total20EditText).
3. Установите значения свойств Id и Text для каждого компонента.

Шаг 11. Добавление компонентов в tableRow4

Добавьте компоненты TextView, SeekBar и другой компонент TextView в tableRow4, выполнив следующие действия:

1. В окне Outline перетащите компонент TextView (customTextView) на tableRow4.
2. В окне Outline перетащите компонент SeekBar (customSeekBar) на tableRow4.
3. В окне Outline перетащите компонент TextView (customTipTextView) на tableRow4.
4. Установите значения свойств Id и Text для компонентов TextView.

Шаг 12. Добавление компонентов в tableRow5

Добавьте компонент TextView, EditText, другой компонент TextView и еще один компонент EditText в tableRow5, выполнив следующие действия:

1. В окне Outline перетащите компонент TextView (tipCustomTextView) на tableRow5.
2. В окне Outline перетащите компонент EditText (tipCustomEditText) на tableRow5.
3. В окне Outline перетащите компонент TextView (totalCustomTextView) на tableRow5.
4. В окне Outline перетащите компонент EditText (totalCustomEditText) на tableRow5.
5. Установите значения свойств Id и Text для каждого компонента.

4.4.3. Просмотр созданного макета

Начиная с этого момента графический интерфейс пользователя будет выглядеть так, как показано на рис. 4.4.

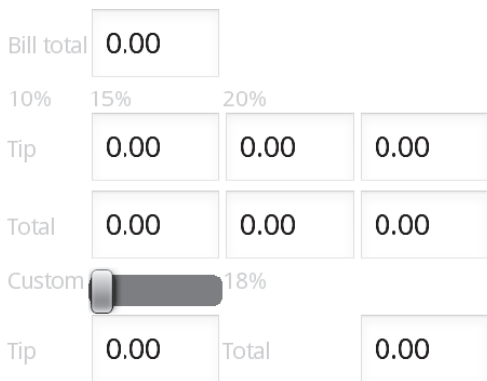


Рис. 4.4. Графический интерфейс пользователя Tip Calculator перед настройкой свойств, отличающихся от свойств Id и Text для каждого компонента

При сравнении с рис. 4.2 заметны следующие различия.

- Компоненты billEditText и customSeekBar еще не охватывают несколько столбцов.
- Текст для всех компонентов TextView выделен светло-серым цветом, что затрудняет его чтение.

- Некоторые из компонентов не «попадают» в свой столбец. Например, компоненты TextView, имеющие заголовки 10 %, 15 % и 20 %, находятся в столбце tableRow1, а компонент TextView с заголовком 18 % — в столбце tableRow4. Последний компонент автоматически «попадет» в свой столбец, если объект customSeekBar будет охватывать два столбца.
- Большая часть текста на рис. 4.2 выровнена по центру или по правому краю, а текст на рис. 4.4 выровнен по левому краю.

4.4.4. Завершение проекта путем настройки компонентов

В результате выполнения дальнейших шагов мы завершим разработку приложения путем настройки его свойств.

Шаг 13. Изменение свойства Text color для всех элементов TextView

В окне Outline можно одновременно выбрать несколько компонентов. Для этого нужно щелкать на каждом компоненте при нажатой клавише Ctrl (либо Control). После завершения выделения в окне Properties отобразятся только общие свойства для выделенных компонентов. При изменении значения свойства для нескольких выделенных компонентов значение свойства изменится для каждого выделенного компонента. Чтобы улучшить читаемость текста для компонентов TextView, измените цвет текста на черный. Для этого нужно одновременно изменить свойство Text color для всех компонентов TextView, выполнив следующие действия:

1. Удерживая клавишу Ctrl (или Control), щелкайте на каждом компоненте TextView для его выделения.
2. В окне Properties найдите свойство Text color и присвойте ему значение #000.

Шаг 14. Перемещение компонентов TextView с заголовками 10 %, 15 % и 20 % в требуемые столбцы

Как показано на рис. 4.2, заголовки столбцов 10 %, 15 % и 20 % находятся во втором, третьем и четвертом столбцах соответственно. По умолчанию при добавлении компонентов в TableRow первый компонент помещается в первый столбец, второй компонент — во второй столбец и т. д. Чтобы переместить компонент в другой столбец, укажите номер столбца компонента.

К сожалению, это свойство по умолчанию не отображается в окне Properties. Поэтому для указания столбца компонента придется изменить соответствующий ему XML-код.

1. Чтобы перейти к просмотру XML-разметки, выберите вкладку main.xml в окне редактора Visual Layout Editor.
2. Выберите элемент <TextView> с атрибутом android:id, имеющим значение "@+id/tenTextView".
3. В открывающий XML-тег компонента TextView добавьте следующую пару «атрибут/значение»:

```
android:layout_column="1"
```

В результате выполнения этой операции компонент TextView с заголовком 10 % переместится во второй столбец (нумерация столбцов будет начинаться с 0). При этом остальные компоненты строки автоматически переместятся в последующие столбцы. Если нужно пропустить другие столбцы, установите значение компонента android:layout_column для каждого компонента в строке таким образом, чтобы определить каждый столбец, в котором будет отображаться компонент. Если в XML-код будет добавлен атрибут вручную, атрибут и его значение будут отображаться в разделе Misc окна Properties.

Шаг 15. Центрирование текста в компонентах TextView объекта tableRow1 и в компонентах EditText объектов tableRow2, tableRow3 и tableRow5, а также установка размера шрифта для компонентов EditText

Как показано на рис. 4.2, текст, отображаемый во многих компонентах, является центрированным. Для выполнения центрирования текста, отображаемого в компонентах, применяется свойство Gravity. Вернитесь обратно на вкладку Graphical Layout в окне Visual Layout Editor и выполните следующие действия:

1. В окне Outline выберите три компонента TextView объекта tableRow1.
2. В окне Properties свойству Gravity присвойте значение center.
3. Выберите все компоненты EditText, относящиеся к объектам tableRow2, tableRow3 и tableRow5.
4. В окне Properties свойству Gravity присвойте значение center.
5. Свойству Text size присвойте значение 14sp. В результате будет уменьшен заданный по умолчанию размер шрифта в компонентах EditText, что позволит отображать больше цифр без переноса текста.

Шаг 16. Охват нескольких столбцов путем настройки компонентов billEditText и customSeekBar

На рис. 4.2 компонент billEditText занимает столбцы 1–3, а компонент customSeekBar — столбцы 1–2. Атрибут, определяющий охват нескольких столбцов, можно добавить непосредственно в XML-код:

1. Чтобы просмотреть разметку макета, выберите вкладку main.xml в окне редактора Visual Layout Editor.
2. Выберите элемент <EditText> с атрибутом android:id, который имеет значение "@+id/billEditText".
3. В открывающий XML-тег EditText добавьте следующую пару «атрибут/значение»:

```
android:layout_span="3"
```

4. Выберите элемент <SeekBar>.
5. В открывающий XML-тег SeekBar добавьте следующую пару «атрибут/значение»:

```
android:layout_span="2"
```

Теперь компонент billEditText будет охватывать столбцы 1–3, а компонент customSeekBar — столбцы 1–2.

Шаг 17. Выравнивание компонентов TextView по правому краю

Компоненты TextView в столбце 0 выровнены по правому краю подобно компоненту TextView, который выровнен по правому краю в третьем столбце объекта tableRow5. Также каждый из элементов TextView имеет отступ справа, равный 5dp. Этот отступ отделяет данный компонент от элемента управления, находящегося справа от него.

1. Вернитесь обратно на вкладку Graphical Layout в окне редактора Visual Layout Editor.
2. В окне Outline выберите все компоненты TextView в столбце 0, а в последней строке — второй компонент TextView.
3. Свойству Gravity присвойте значение right, а свойству Padding right — значение 5dp.

Шаг 18. Центрирование по вертикали компонентов TextView в tableRow4

А теперь настроим свойство Gravity, чтобы выровнять по вертикали компоненты TextView, относящиеся к объекту tableRow4:

1. В окне Outline выберите компонент customTextView объекта tableRow4.
2. Выберите свойство Gravity и щелкните на кнопке с многоточием, которая находится справа от значения свойства, чтобы отобразить список возможных значений свойства Gravity.
3. Выберите значение свойства center_vertical. Следует установить значения right и center_vertical.
4. Чтобы применить значение, щелкните на кнопке OK.
5. В окне Outline выберите компонент customTipTextView, относящийся к объекту tableRow4.
6. Свойству Gravity присвойте значение center_vertical.
7. Для применения значения щелкните на кнопке OK.
8. В окне Outline выберите два компонента TextView объекта tableRow4, свойствам Layout height присвойте значения match_parent, а свойству Padding bottom присвойте значение 5dp. В результате два компонента TextView будут иметь ту же высоту, что и компонент SeekBar, а с помощью свойства Gravity текст будет выравниваться по вертикали вместе с компонентом SeekBar. Свойству Padding bottom также будет присвоено значение SeekBar. Благодаря настройке значения этого свойства для свойства TextView текст будет выровнен относительно компонента SeekBar.
9. И наконец, присвоим свойству Padding left объекта customTipTextView значение 5dp, чтобы визуально отделить компонент TextView от компонента SeekBar.

Шаг 19. Установка свойств Progress Property и Padding объекта customSeekBar

Чтобы завершить создание графического интерфейса пользователя, присвойте значения свойствам Progress, Padding left и Padding right компонента SeekBar. Сначала настройте

положение ползунка SeekBar, чтобы представить значение 18 %, которое отображается в TextView справа от SeekBar. Также добавьте небольшой отступ слева и справа от SeekBar. Если ползунок перемещается в крайнее левое или крайнее правое положение компонента SeekBar (представляя значения 0 и 100 соответственно), его довольно трудно захватывать, поскольку недостаточно свободного места между компонентом SeekBar и другими компонентами, находящимися слева и справа от него.

1. В окне Outline выберите компонент customSeekBar.
2. Свойству Progress присвойте значение 18.
3. Свойствам Padding left и Padding right присвойте значение 8dp.
4. Свойству Padding bottom присвойте значение 5dp, чтобы отделить с помощью отступа последнюю строку компонентов.
5. Свойству Focusable присвойте значение false. После этого в случае изменения пользователем значения SeekBar компонент billEditText будет сохранять фокус. Это полезно в том случае, если нужно отображать на экране клавиатуру (для устройств, отображающих программную клавиатуру).

Шаг 20. Предотвращение возможности изменения пользователем результатов вычислений, отображаемых компонентом EditText

За исключением компонента billEditText, находящегося в верхней части графического интерфейса пользователя, все другие компоненты EditText приложения применяются лишь для отображения результатов вычислений. Поэтому пользователь не может изменять текст, соответствующий этим компонентам. Контролировать получение фокуса компонента EditText пользователем можно путем установки свойства Focusable. Можно также запретить длинное нажатие на компоненте EditText, а также предотвратить отображение курсора в области EditText. Это приведет к блокированию доступа пользователя к тексту. Настройте следующие параметры:

1. В окне Outline выберите все компоненты EditText за исключением компонента billEditText.
2. Свойствам Focusable, Long clickable и Cursor visible присвойте значения false.

Шаг 21. Указание типа числовой клавиатуры с помощью billEditText

Можно настроить параметры, обеспечивающие ввод пользователем чисел с плавающей запятой в компонент billEditText. Выполните следующие действия:

1. В окне Outline выберите компонент billEditText.
2. Свойству Input type присвойте значение numberDecimal.

Шаг 22. Настройка значений «веса» макета для различных компонентов

Свойство Layout weight компонента определяет его важность (вес) относительно других компонентов. По умолчанию свойству Layout weight для всех компонентов присваивается значение 0. Это свойство определяет относительную величину данного компонента по отношению к другим компонентам. В данном случае свойству Layout weight присваивается 1 для всех компонентов за исключением компонентов TextView, находящихся

в левом столбце. Если макет растягивается для заполнения экрана, компоненты TextView, находящиеся в левом столбце экрана, занимают область, ширина которой соответствует ширине наиболее широкого компонента TextView в этом столбце. Другие компоненты со свойством Layout weight, которому присвоено значение 1, растягиваются для заполнения оставшейся области экрана. При этом пространство экрана делится равномерно. Если свойству Layout weight для компонента строки присвоено значение 2, этот компонент будет занимать в два раза больше места, чем компонент со свойством Layout weight, значение которого равно 1 (в этой же строке).

На этом разработка графического интерфейса пользователя завершена. В следующем разделе будет представлена XML-разметка, сгенерированная редактором Visual Layout Editor. В разделе 4.5 приводится код приложения.

4.4.5. Завершенная XML-разметка GUI приложения Tip Calculator

Теперь разработанный вами GUI будет выглядеть так, как показано на рис. 4.2. В листинге 4.1 представлена завершенная XML-разметка графического интерфейса пользователя приложения Tip Calculator. Сгенерированный XML-код был переформатирован, а также добавлены комментарии, облегчающие читабельность. Выделены области кода, определяющие новые функции GUI, которые были рассмотрены в разделах 4.4.2 и 4.4.4.

Листинг 4.1. XML-разметка приложения Tip Calculator

```
<?xml version="1.0" encoding="utf-8"?>
<!-- main.xml -->
<! – XML-разметка приложения Tip Calculator -->

<TableLayot xmlns:android=http://schemas.android.com/apk/res/android
   android:layout_width="match_parent"
     android:layout_height="match_parent"
   android:background="#FFF" android:id="@+id/tableLayout"
  android:stretchColumns="1,2,3" android:padding="5dp">

<!-- tableRow0 -->
<TableRow android:layout_height="wrap_content"
   android:layout_width="match_parent" android:id="@+id/tableRow0">
      <TextView android:id="@+id/billTextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/billTotal" android:textColor="#000"
         android:gravity="right" android:paddingRight="5dp"></TextView>
      <EditText android:layout_width="wrap_content"
         android:id="@+id/billEditText"
         android:layout_height="wrap_content" android:layout_span="3"
         android:inputType="numberDecimal" android:layout_weight="1">
      </EditText>
</TableRow>

<!-- tableRow1 -->
<TableRow android:layout_height="wrap_content"
```

```
   android:layout_width="match_parent" android:id="@+id/tableRow1">
      <TextView android:id="@+id/tenTextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" android:text="10 %"
         android:textColor="#000" android:layout_column="1"
         android:gravity="center" android:layout_weight="1"></TextView>
      <TextView android:id="@+id/fifteenTextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" android:text="15 %"
         android:textColor="#000" android:gravity="center"
         android:layout_weight="1"></TextView>
      <TextView android:id="@+id/twentyTextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" android:text="20 %"
         android:textColor="#000" android:gravity="center"
         android:layout_weight="1"></TextView>
</TableRow>

<!-- tableRow2 -->
<TableRow android:layout_height="wrap_content"
   android:layout_width="match_parent" android:id="@+id/tableRow2">
      <TextView android:id="@+id/tipTextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/tip" android:textColor="#000"
         android:gravity="right" android:paddingRight="5dp"></TextView>
   <EditText android:layout_width="wrap_content"
      android:id="@+id/tip10EditText"
      android:layout_height="wrap_content" android:text="@string/zero"
      android:gravity="center" android:focusable="false"
      android:layout_weight="1" android:textSize="14sp"
      android:cursorVisible="false" android:longClickable="false">
   </EditText>
   <EditText android:layout_width="wrap_content"
      android:id="@+id/tip15EditText"
      android:layout_height="wrap_content" android:text="@string/zero"
      android:gravity="center" android:focusable="false"
      android:layout_weight="1" android:textSize="14sp"
      android:cursorVisible="false" android:longClickable="false">
   </EditText>
   <EditText android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:id="@+id/tip20EditText" android:text="@string/zero"
      android:gravity="center" android:focusable="false"
      android:layout_weight="1" android:textSize="14sp"
      android:cursorVisible="false" android:longClickable="false">
   </EditText>
</TableRow>

<!-- tableRow3 -->
<TableRow android:layout_height="wrap_content"
```

продолжение ➪

Листинг 4.1 *(продолжение)*

```
  android:layout_width="match_parent" android:id="@+id/tableRow3">
  <TextView android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:id="@+id/totalTextView" android:text="@string/total"
     android:textColor="#000" android:gravity="right"
     android:paddingRight="5dp"></TextView>
  <EditText android:layout_width="wrap_content"
     android:text="@string/zero" android:layout_height="wrap_content"
     android:id="@+id/total10EditText" android:gravity="center"
     android:focusable="false" android:layout_weight="1"
     android:textSize="14sp" android:cursorVisible="false"
     android:longClickable="false"></EditText>
  <EditText android:layout_width="wrap_content"
     android:text="@string/zero" android:layout_height="wrap_content"
     android:id="@+id/total15EditText" android:gravity="center"
     android:focusable="false" android:layout_weight="1"
     android:textSize="14sp" android:cursorVisible="false"
     android:longClickable="false"></EditText>
  <EditText android:layout_width="wrap_content"
     android:text="@string/zero" android:layout_height="wrap_content"
     android:id="@+id/total20EditText" android:gravity="center"
     android:focusable="false" android:layout_weight="1"
     android:textSize="14sp" android:cursorVisible="false"
     android:longClickable="false"></EditText>
</TableRow>

<!-- tableRow4 -->
<TableRow android:layout_height="wrap_content"
   android:layout_width="match_parent" android:id="@+id/tableRow4">
   <TextView android:id="@+id/customTextView"
      android:layout_width="wrap_content" android:text="@string/custom"
      android:textColor="#000" android:paddingRight="5dp"
      android:gravity="right|center_vertical"
      android:layout_height="match_parent" android:paddingBottom="5dp"
      android:focusable="false"></TextView>
   <SeekBar android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:id="@+id/customSeekBar" android:layout_span="2"
      android:progress="18" android:paddingLeft="8dp"
      android:paddingRight="8dp" android:paddingBottom="5dp"
      android:layout_weight="1"></SeekBar>
   <TextView android:id="@+id/customTipTextView"
      android:layout_width="wrap_content" android:text="18 %"
      android:textColor="#000" android:gravity="center_vertical"
      android:layout_height="match_parent" android:paddingLeft="5dp"
      android:paddingBottom="5dp" android:focusable="false"
      android:layout_weight="1"></TextView>
</TableRow>

<!-- tableRow5 -->
```

```
<TableRow android:layout_height="wrap_content"
   android:layout_width="match_parent" android:id="@+id/tableRow5">
   <TextView android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/tipCustomTextView" android:text="@string/tip"
      android:textColor="#000" android:gravity="right"
      android:paddingRight="5dp"></TextView>
   <EditText android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/tipCustomEditText" android:text="@string/zero"
      android:gravity="center" android:focusable="false"
      android:layout_weight="1" android:textSize="14sp"
      android:cursorVisible="false" android:longClickable="false">
   </EditText>
   <TextView android:id="@+id/totalCustomTextView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="@string/total"
      android:textColor="#000" android:gravity="right"
      android:paddingRight="5dp" android:layout_weight="1"></TextView>
   <EditText android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:id="@+id/totalCustomEditText" android:text="@string/zero"
      android:gravity="center" android:focusable="false"
      android:layout_weight="1" android:textSize="14sp"
      android:cursorVisible="false" android:longClickable="false">
   </EditText>
</TableRow>
</TableLayout>
```

4.4.6 Файл strings.xml

В листинге 4.2 приведены строковые ресурсы, используемые в листинге 4.1.

Листинг 4.2. Строковые ресурсы, которые хранятся в файле strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">Tip Calculator</string>
   <string name="billTotal">Bill total</string>
   <string name="tip">Tip</string>
   <string name="total">Total</string>
   <string name="custom">Custom</string>
   <string name="zero">0.00</string>
</resources>
```

4.5. Включение дополнительных функций в приложение

В листингах 4.3–4.11 приведена реализация приложения Tip Calculator в единственном классе TipCalculator. Этот класс вычисляет чаевые на основе процентной ставки, равной 10 %, 15 %, 20 %, а также на базе пользовательской процентной ставки. Чаевые

вычисляются на основании выставленного счета, затем вычисленные чаевые добавляются к выставленному счету, образуя счет к оплате (итоговый счет).

Операторы package и import

В листинге 4.3 приводится код операторов package и import, который включен в класс TipCalculator.java. Оператор package, находящийся в строке 3, указывает на то, что класс в этом файле является частью пакета package com.deitel.tipcalculator. Эта строка была вставлена при создании проекта на шаге 1 в разделе 4.4.

Листинг 4.3. Операторы package и import, входящие в состав класса TipCalculator

```
 1 // TipCalculator.java
 2 // Вычисляет чаевые на основе счета с 5, 10, 15 или
   // введенной пользователем процентной ставки.
 3 package com.deitel.tipcalculator;
 4
 5 import android.app.Activity;
 6 import android.os.Bundle;
 7 import android.text.Editable;
 8 import android.text.TextWatcher;
 9 import android.widget.EditText;
10 import android.widget.SeekBar;
11 import android.widget.SeekBar.OnSeekBarChangeListener;
12 import android.widget.TextView;
13
```

Операторы import, находящиеся в строках 5–12, импортируют различные интерфейсы и классы, используемые приложением.

- Класс Activity, относящийся к пакету android.app (строка 5), поддерживает *базовые методы жизненного цикла*, используемые в приложении (эти методы будут рассмотрены ниже).
- Класс Bundle из пакета android.os (строка 6) представляет информацию о состоянии приложения. Приложение может сохранять свое состояние в случае перевода в фоновый режим операционной системой (например, если пользователь запускает другое приложение или принимает телефонный звонок).
- Интерфейс Editable из пакета android.text (строка 7) позволяет изменять содержимое и разметку текста в GUI.
- Интерфейс TextWatcher из пакета android.text (строка 8) служит для реагирования на события, возникающие при взаимодействии пользователя с компонентом EditText.
- Пакет android.widget (строки 9–12) содержит виджеты (то есть компоненты GUI) и макеты, используемые в Android GUI, такие как EditText (строка 9), SeekBar (строка 10) и TextView (строка 12).
- Интерфейс SeekBar.OnSeekBarChangeListener из пакета android.widget (строка 11) предназначен для реагирования на события, возникающие при перемещении ползунка SeekBar.

Класс Activity и жизненный цикл деятельностей приложения Tip Calculator

Приложения Android *не включают главный метод (main method)*. Они могут включать четыре типа компонентов: *деятельности*, *службы*, *провайдеры контента и широковещательные приемники*. Порядок инициализации этих компонентов рассмотрим позднее. В этой главе мы обсуждаем только деятельности. Взаимодействие пользователей с деятельностями осуществляется с помощью представлений (или видов), которые являются компонентами GUI. Каждая отдельная деятельность обычно связана с отдельным экраном приложения.

Класс TipCalculator (см. листинги 4.4–4.11) представляет собой класс Activity приложения Tip Calculator. В следующих главах будут рассмотрены примеры приложений, включающих несколько деятельностей, представляющих отдельные экраны приложения. Класс TipCalculator является результатом расширения (наследуется) класса Activity (строка 15). В процессе создания проекта TipCalculator модуль ADT Plugin генерирует этот класс как подкласс класса Activity и поддерживает оболочку для переопределенного метода onCreate, с помощью которого переопределяется каждый подкласс Activity. Ниже мы рассмотрим этот метод.

Листинг 4.4. Класс TipCalculator является подклассом класса Activity

```
14 // Главный класс Activity для приложения TipCalculator
15 public class TipCalculator extends Activity
16 {
```

На протяжении своего жизненного цикла деятельность может находиться в одном из следующих *состояний*: *активное* (выполняется), *приостановленное* или *остановленное*. Переход между различными состояниями деятельности осуществляется в ответ на различные события.

- *Активная* (или выполняющаяся) деятельность отображается на экране и «владеет фокусом» (то есть выполняется на переднем плане). Именно с этой деятельностью взаимодействует пользователь.
- *Приостановленная* деятельность *отображается* на экране, но не обладает фокусом. Эта деятельность может быть удалена из памяти операционной системой (как правило, с целью освобождения места, требуемого для выполнения другого приложения). Обратите внимание, что в первую очередь из памяти удаляются *остановленные* деятельности.
- *Остановленная* деятельность *не отображается* на экране и будет удалена системой при нехватке памяти.

В процессе перехода деятельности между различными состояниями она принимает вызовы для различных методов жизненного цикла, определенных в классе Activity (developer.android.com/reference/android/app/Activity.html). В приложении Tip Calculator реализованы два метода жизненного цикла: onCreate и onSaveInstanceState. К другим важнейшим методам жизненного цикла относятся: onStart, onPause, onRestart, onResume, onStop и onDestroy. Все эти методы будут рассмотрены в следующих главах.

- Метод onCreate вызывается системой после запуска класса Activity. После вызова этого метода отображается графический интерфейс пользователя, а пользователь получает возможность взаимодействия с классом Activity.
- Метод onSaveInstanceState вызывается системой в случае изменения конфигурации устройства в процессе выполнения приложения (например, при вращении устройства пользователем или выдвижении физической клавиатуры устройства, как в оригинальном смартфоне Motorola Droid). Этот метод может применяться для сохранения информации о состоянии, которое может быть восстановлено в случае вызова метода onCreate (в качестве составной части процесса изменения конфигурации). Если приложение находится в фоновом режиме, обеспечивающем возможность ответа пользователя на телефонные звонки, либо в случае запуска пользователем другого приложения, компоненты GUI приложения будут автоматически сохранять свой контент. В результате обеспечивается возврат к выполнению приложения на переднем плане (эта возможность будет доступной в том случае, если система не «убивает» приложение).

Каждый метод жизненного цикла деятельности, переопределенный пользователем, должен сначала вызывать версию суперкласса этого метода. В *противном случае вызов этого метода приведет к генерированию исключения.*

Переменные класса и экземпляры переменных

В строках 18–32 листинга 4.5 объявляются переменные класса TipCalculator. Большинство из этих переменных представляют собой компоненты EditText, используемые для ввода выставленного счета пользователем, а также для отображения вычисленной суммы чаевых и общего счета (с учетом чаевых). Статические строки (строки 18–19) используются в качестве ключей в парах «ключ/значение» для представления текущего итогового счета и пользовательских процентных ставок чаевых. Пары «ключ/значение» хранятся и выбираются с помощью методов SaveInstanceState и onCreate соответственно (при изменении конфигурации приложения).

Листинг 4.5. Переменные экземпляра класса TipCalculator

```
17 // константы, используемые при сохранении/восстановлении состояния
18 private static final String BILL_TOTAL = "BILL_TOTAL";
19 private static final String CUSTOM_PERCENT = "CUSTOM_PERCENT";
20
21 private double currentBillTotal;      // счет, вводимый пользователем
22 private int currentCustomPercent;     //  % чаевых, выбранный SeekBar
23 private EditText tip10EditText;       // 10%-чаевые
24 private EditText total10EditText;     // общий счет, включая 10%-чаевые
25 private EditText tip15EditText;       // 15%-чаевые
26 private EditText total15EditText;     // общий счет, включая 15%-чаевые
27 private EditText billEditText;        // ввод счета пользователем
28 private EditText tip20EditText;       // 20%-чаевые
29 private EditText total20EditText;     // общий счет, включая 20%-чаевые
30 private TextView customTipTextView;   //  % пользовательских чаевых
31 private EditText tipCustomEditText;   // пользовательские чаевые
32 private EditText totalCustomEditText; // общий счет
33                                       // с пользовательскими чаевыми
```

Выставленный счет, введенный пользователем в EditText billEditText, считывается и хранится в виде строки в компоненте currentBillTotal. При этом требуется выполнение определенного преобразования, суть которого будет объяснена позже. Пользовательская процентная ставка чаевых, выбираемая пользователем с помощью ползунка Seekbar (целое число, находящееся в диапазоне 0–100), будет сохранена в currentCustomPercent. Это значение обычно умножается на 0,01 для преобразования в тип данных double, используемый в вычислениях. Сумма пользовательских чаевых и общий счет, включая пользовательские чаевые, присваиваются компонентам tipCustomEditText и totalCustomEditText соответственно. В строке 30 объявляется переменная TextView, в которой хранится процентная ставка пользовательских чаевых, соответствующая отображаемой позиции ползунка SeekBar (см. значение 18 % на рис. 4.1, *а*).

Фиксированные процентные ставки чаевых, равные 10 %, 15 % и 20 %, а также итоговые счета (с учетом чаевых) отображаются в EditText. Сумма чаевых, подсчитанная с учетом ставки 10 %, и сумма итогового счета (вместе с чаевыми, подсчитанными по ставке 10 %) хранятся в переменных tip10EditText и total10EditText соответственно. Сумма чаевых, подсчитанная с учетом ставки 15 %, и сумма итогового счета (вместе с чаевыми, подсчитанными по ставке 15 %) хранятся с помощью переменных tip15EditText и total15EditText соответственно. Сумма чаевых, подсчитанная с учетом ставки 20 %, и сумма итогового счета (вместе с чаевыми, подсчитанными по ставке 20 %) хранятся с помощью переменных tip20EditText и total20EditText соответственно.

Переопределение метода OnCreate из класса Activity

Метод onCreate (см. листинг 4.6) генерируется автоматически при создании проекта приложения и вызывается системой после запуска класса Activity. Метод onCreate обычно инициализирует переменные экземпляра класса Activity и компоненты GUI. Этот метод упрощен до предела, поэтому приложение загружается быстро. Фактически в случае, если загрузка приложения занимает более пяти секунд, операционная система отображает диалоговое окно ANR (Application Not Responding, приложение не отвечает). В этом окне отображается параметр, с помощью которого пользователь может принудительно завершить приложение. Операции по инициализации, занимающие много времени, следует выполнять в фоновом процессе, а не с помощью метода onCreate.

Листинг 4.6. Переопределение метода onCreate класса Activity

```
34 // Вызывается при первом создании класса activity.
35 @Override
36 public void onCreate(Bundle savedInstanceState)
37 {
38    super.onCreate(savedInstanceState); // вызов версии суперкласса
39    setContentView(R.layout.main);      // «раздувание» GUI
40
41    // Приложение запущено впервые или восстановлено из памяти?
42    if ( savedInstanceState == null )   // приложение запущено впервые
43    {
44       currentBillTotal = 0.0;    // инициализация суммы счета нулем
45       currentCustomPercent = 18; // инициализация пользовательских
                                    // чаевых значением 18 %
46    }  // конец структуры if
```

продолжение ☞

Листинг 4.6 *(продолжение)*

```
47     else // приложение восстановлено из памяти
48     {
49        // инициализация суммы счета сохраненной в памяти суммой
50        currentBillTotal = savedInstanceState.getDouble(BILL_TOTAL);
51
52        // инициализация пользовательских чаевых сохраненным
          // процентом чаевых
53        currentCustomPercent =
54           savedInstanceState.getInt(CUSTOM_PERCENT);
55 }        // конец структуры else
56
57     // ссылки на чаевые 10 %, 15 %, 20 % и итоговые EditTexts
58     tip10EditText = (EditText) findViewById(R.id.tip10EditText);
59     total10EditText = (EditText) findViewById(R.id.total10EditText);
60     tip15EditText = (EditText) findViewById(R.id.tip15EditText);
61     total15EditText = (EditText) findViewById(R.id.total15EditText);
62     tip20EditText = (EditText) findViewById(R.id.tip20EditText);
63     total20EditText = (EditText) findViewById(R.id.total20EditText);
64
65     // TextView, отображающий процент пользовательских чаевых
66     customTipTextView = (TextView) findViewById(R.id.customTipTextView);
67
68     // пользовательские чаевые и итоговые EditTexts
69     tipCustomEditText = (EditText) findViewById(R.id.tipCustomEditText);
70     totalCustomEditText =
71        (EditText) findViewById(R.id.totalCustomEditText);
72
73     // получение billEditText
74     billEditText = (EditText) findViewById(R.id.billEditText);
75
76     // billEditTextWatcher обрабатывает событие onTextChanged
       // из billEditText
77     billEditText.addTextChangedListener(billEditTextWatcher);
78
79     // получение SeekBar, используемого для подсчета суммы
       // пользовательских чаевых
80     SeekBar customSeekBar = (SeekBar) findViewById(R.id.customSeekBar);
81     customSeekBar.setOnSeekBarChangeListener(customSeekBarListener);
82   } // конец метода onCreate
83
```

Во время выполнения приложения пользователь может изменить конфигурацию устройства путем его поворота либо выдвижением физической клавиатуры. Приложение без проблем переключается в различные выбранные конфигурации. Если система вызывает метод onCreate, он передает значение Bundle параметру savedInstanceState. Это значение содержит сохраненное состояние деятельности (при наличии подобного состояния). Обычно информация, относящаяся к этому состоянию, сохраняется с помощью метода onSaveInstanceState из класса Activity (см. листинг 4.9). (В данном случае использовался метод savedInstanceState, определенный в строках 42–55.) В строке 38

вызывается метод onCreate суперкласса, который используется при переопределении любого метода из класса Activity.

В процессе создания графического интерфейса пользователя и добавления ресурсов (например, строк в файл strings.xml или GUI-компонентов в файл main.xml) в приложение инструменты плагина ADT генерируют класс R. Этот класс содержит вложенные статические классы, представляющие каждый тип ресурса, находящийся в папке проекта res. Класс R находится в папке проекта gen, в которой находятся сгенерированные файлы исходного кода. В классах, вложенных в класс R, инструменты создают константы static final intst, с помощью которых можно ссылаться на эти ресурсы программным образом из кода приложения (методы программного доступа будут обсуждаться далее). Рассмотрим некоторые классы, вложенные в класс R:

- Класс drawable. Этот класс включает все элементы drawable, например изображения, которые находятся в различных папках drawable, расположенных в папке приложения res.
- Класс id. Данный класс включает константы для GUI-компонентов, используемые в файлах XML-разметки.
- Класс layout. Здесь находятся константы, которые представляют файлы разметки проекта (например, main.xml).
- Класс string. В этом классе находятся константы, используемые для определения строк в файле strings.xml.

В результате вызова setContentView (строка 39) принимается значение константы R.layout.main, с помощью которой выбирается XML-файл, который представляет GUI деятельности. В данном случае константа представляет файл main.xml. Метод setContentView использует эту константу для загрузки соответствующего XML-документа, который потом анализируется и преобразуется в графический интерфейс пользователя приложения. Этот процесс известен под названием *«раздувание»* GUI.

В строках 42–55 определяется, было ли приложение запущено в первый раз либо восстановлено после изменения конфигурации. Если значение переменной savedInstanceState будет null (строка 42), приложение запускается первый раз. В этом случае в строках 44–45 переменные currentBillTotal и currentCustomPercent инициализируются значениями, необходимыми в случае первой загрузки приложения. Если же приложение было восстановлено из памяти, в строке 50 вызывается метод getString объекта savedInstanceState, с помощью которого считывается сумма сохраненного итогового счета (в виде значения double). В строках 53–54 вызывается метод getInt объекта savedInstanceState, позволяющий получить доступ к сохраненной процентной ставке пользовательских чаевых (в виде значения int).

После «раздувания» XML-разметки с помощью метода findViewById класса Activity обеспечивается получение ссылок на отдельные виджеты. Этот метод получает константу типа int для специфического представления (GUI-компонент) и возвращает ссылку на нее. Название каждой константы из компонента GUI в классе R.id определяется с помощью атрибута android:id компонента GUI из файла main.xml. Например, значение константы billEditText будет R.id.billEditText.

В строках 58–63 обеспечивается получение ссылок на шесть компонентов EditText, в которых хранятся вычисленные чаевые (с процентными ставками 10 %, 15 % и 20 %),

а также итоговые суммы по счетам (с учетом чаевых). В строке 66 содержится ссылка на компонент TextView, которая обновляется в случае изменения пользователем процентной ставки пользовательских чаевых. В строках 69–71 находятся ссылки на компоненты EditText, в которых отображаются вычисленные пользовательские чаевые и итоговые суммы по счетам.

В строке 74 находится ссылка на billEditText, а в строке 77 вызывается метод addTextChangedListener. Этот метод регистрирует объект TextChangedListener, который отвечает на события, сгенерированные в случае изменения пользователем текста billEditText. Код этого объекта listener приведен в листинге 4.11.

В строке 80 находится ссылка на объект customSeekBar, а в строке 81 вызывается метод setOnSeekBar-ChangeListener, регистрирующий объект OnSeekBarChangeListener. Этот объект отвечает на события, которые генерируются в случае перемещения пользователем ползунка customSeekBar, с помощью которого изменяется процентная ставка пользовательских чаевых. Код этого объекта listener приводится в листинге 4.10.

Метод updateStandard из класса TipCalculator

Метод updateStandard (см. листинг 4.7) обновляет значения EditText, в которых находятся чаевые, вычисленные в соответствии с процентными ставками 10 %, 15 % и 20 %, а также итоговые чаевые. Обновление осуществляется при каждом изменении итогового счета. Метод использует значение currentBillTotal для вычисления сумм чаевых и итогового счета при выборе процентной ставки 10 % (строки 88–95), 15 % (строки 98–106) и 20 % (строки 109–116). Метод static format из класса String применяется для преобразования сумм чаевых и счетов в строки, которые отображаются соответствующими компонентами EditText.

Листинг 4.7. Метод updateStandard из класса TipCalculator вычисляет и отображает чаевые и итоговые счета на основе стандартных процентных ставок чаевых (10 %, 15 % и 20 %)

```
 84 // вычисляет чаевые, хранящиеся в EditTexts, по ставкам 10, 15 и 20 %
 85 private void updateStandard()
 86 {
 87    // вычисляет итоговый счет, включающий чаевые со ставкой 10 %
 88    double tenPercentTip = currentBillTotal * .1;
 89    double tenPercentTotal = currentBillTotal + tenPercentTip;
 90
 91    // настройка текста tipTenEditText в соответствии с tenPercentTip
 92    tip10EditText.setText(String.format(" %.02f", tenPercentTip));
 93
 94    // настройка текста totalTenEditText в соответствии с tenPercentTotal
 95    total10EditText.setText(String.format(" %.02f", tenPercentTotal));
 96
 97    // вычисление общего итога с чаевыми 15 %
 98    double fifteenPercentTip = currentBillTotal * .15;
 99   double fifteenPercentTotal = currentBillTotal + fifteenPercentTip;
100
101    // настройка текста tipFifteenEditText в соответствии
       // с  fifteenPercentTip
```

```
   tip15EditText.setText(String.format(" %.02f", fifteenPercentTip));

   // настройка текста totalFifteenEditText в соответствии
   // с fifteenPercentTotal
   total15EditText.setText(
   String.format(" %.02f", fifteenPercentTotal));

   // вычисление общего итога с чаевыми 20 %
   double twentyPercentTip = currentBillTotal * .20;
   double twentyPercentTotal = currentBillTotal + twentyPercentTip;

   // настройка текста tipTwentyEditText в соответствии
   //с twentyPercentTip
   tip20EditText.setText(String.format(" %.02f", twentyPercentTip));

   // настройка текста totalTwentyEditText в соответствии
   // с  twentyPercentTotal
   total20EditText.setText(String.format(" %.02f", twentyPercentTotal));
} // конец метода updateStandard

```

Метод updateCustom из класса TipCalculator

Метод updateCustom (см. листинг 4.8) обновляет значения компонентов EditText, содержащих значения пользовательских чаевых и итогового сета. При этом используется процентная ставка, выбранная пользователем с помощью customSeekBar. В строке 123 настраивается текст customTipTextView таким образом, чтобы достигалось соответствие с положением ползунка SeekBar. В строках 126–127 вычисляется значение customTipAmount. В строке 130 вычисляется customTotalAmount. В строках 133–135 выполняется преобразование customTipAmount и customTotalAmount в строки, которые отображаются в tipCustomEditText и totalCustomEditText соответственно.

Листинг 4.8. Метод updateCustom из класса TipCalculator вычисляет и отображает чаевые и итоговый счет в случае пользовательской ставки чаевых, выбираемой пользователем с помощью customSeekBar

```
119 // обновляет компоненты EditText, включающие пользовательские
    // чаевые и итоги
120 private void updateCustom()
121 {
122    // настройка текста customTipTextView в соответствии с положением
       // SeekBar
123    customTipTextView.setText(currentCustomPercent + " %");
124
125    // вычисление суммы пользовательских чаевых
126    double customTipAmount =
127       currentBillTotal * currentCustomPercent * .01;
128
129    // вычисление итогового счета, включая пользовательские чаевые
130    double customTotalAmount = currentBillTotal + customTipAmount;
131
```

продолжение ⇨

Листинг 4.8 *(продолжение)*

```
      // отображение суммы чаевых и итогового счета
      tipCustomEditText.setText(String.format(" %.02f", customTipAmount));
      totalCustomEditText.setText(
      String.format(" %.02f", customTotalAmount));
   }  // конец метода updateCustom

```

Переопределение метода onSaveInstanceState из класса Activity

В строках 139–146 (см. листинг 4.9) переопределяется метод onSaveInstanceState из класса Activity. Этот метод вызывается системой в случае изменения конфигурации устройства при выполнении приложения (например, при вращении устройства пользователем или выдвижении клавиатуры устройства). Чтобы сгенерировать этот метод в Eclipse, щелкните правой кнопкой мыши в области исходного кода и в контекстном меню выберите команды Source ▸ Override/Implement Methods... (Исходный код ▸ Переопределение/реализация методов). В появившемся диалоговом окне отображаются методы, которые могут быть переопределены или реализованы в классе. Установите флажок onSaveInstanceState, укажите место, в которое IDE включит код класса, и щелкните на кнопке OK для создания оболочки метода.

Листинг 4.9. Переопределение метода onSaveInstanceState из класса Activity для сохранения состояния при изменении конфигурации приложения

```
138  // сохранение значений billEditText и customSeekBar
139  @Override
140  protected void onSaveInstanceState(Bundle outState)
141  {
142     super.onSaveInstanceState(outState);
143
144     outState.putDouble( BILL_TOTAL, currentBillTotal );
145     outState.putInt( CUSTOM_PERCENT, currentCustomPercent );
146  }  // конец метода onSaveInstanceState
147
```

В разрабатываемом в этой главе приложении сначала вызывается метод onSaveInstanceState суперкласса, потом сохраняются пары «ключ/значение» в Bundle, которые потом передаются методу. В строке 144 сохраняется текущая сумма итогового счета, а в строке 145 сохраняется процентная ставка пользовательских чаевых (соответствует текущему положению ползунка SeekBar). Эти значения используются методом onCreate, который вызывается для восстановления приложения после изменения конфигурации. При рассмотрении других приложений из этой книги мы изучим множество других методов жизненного цикла Activity. Эти методы описаны на веб-сайте:

bit.ly/ActivityLifeCycle.

Анонимный внутренний класс, реализующий интерфейс OnSeekBarChangeListener

В строках 149–171 (см. листинг 4.10) создается анонимный объект внутреннего класса customSeekBarListener, который реагирует на события customSeekBar. Дополнительные

сведения, касающиеся анонимных внутренних классов, можно найти на странице Oracle Java Tutorial bit.ly/AnonymousInnerClasses.

В строке 81 регистрируется customSeekBarListener в качестве объекта, выполняющего обработку событий и относящегося к customSeekBar. В строках 153–170 реализованы методы интерфейса OnSeekBarChangeListener.

Листинг 4.10. Анонимный внутренний класс, реализующий интерфейс OnSeekBarChangeListener в ответ на события, вызываемые customSeekBar

```
148 // вызывается при изменении пользователем положения ползунка SeekBar
149 private OnSeekBarChangeListener customSeekBarListener =
150    new OnSeekBarChangeListener()
151 {
152    // обновление currentCustomPercent, потом вызов updateCustom
153    @Override
154    public void onProgressChanged(SeekBar seekBar, int progress,
155       boolean fromUser)
156    {
157       // присваивание currentCustomPercent положения ползунка SeekBar
158       currentCustomPercent = seekBar.getProgress();
159       updateCustom(); // обновление EditTexts пользовательскими
                         // чаевыми и итоговыми счетами
160    } // конец метода onProgressChanged
161
162    @Override
163    public void onStartTrackingTouch(SeekBar seekBar)
164    {
165 }  // конец метода onStartTrackingTouch
166
167    @Override
168    public void onStopTrackingTouch(SeekBar seekBar)
169    {
170 }  // конец метода onStopTrackingTouch
171 }; // конец OnSeekBarChangeListener
172
```

Переопределение метода onProgressChanged интерфейса OnSeekBarChangeListener

В строках 153–160 переопределяется метод onProgressChanged. В строке 158 метод getProgress объекта SeekBar возвращает целочисленное значение, находящееся в диапазоне 0–100. Это значение соответствует положению ползунка объекта SeekBar и присваивается переменной currentCustomPercent. В строке 159 вызывается метод updateCustom, который используется в customCurrentPercent для вычисления и отображения пользовательских чаевых и итогового счета.

Переопределение методов onStartTrackingTouch и onStopTrackingTouch интерфейса OnSeekBarChangeListener

В Java нужно переопределять каждый реализованный метод интерфейса. И хотя в создаваемом приложении не используются эти методы интерфейса, чтобы выполнить правила Java, создадим пустую оболочку для каждого метода (строки 162–170).

Анонимный внутренний класс, реализующий интерфейс TextWatcher

В строках 174–206 листинга 4.11 создается объект внутреннего класса billEditTextWatcher, реагирующий на события, порождаемые billEditText. В строке 77 регистрируется billEditTextWatcher для прослушивания событий, порождаемых billEditText. В строках 177–205 реализуются методы интерфейса TextWatcher.

Листинг 4.11. Анонимный внутренний класс, реализующий интерфейс TextWatcher, который используется для реагирования на события, порождаемые billEditText

```
173 // объект обработки событий, реагирующий на события billEditText
174 private TextWatcher billEditTextWatcher = new TextWatcher()
175 {
176    // вызывается после ввода пользователем числа
177    @Override
178    public void onTextChanged(CharSequence s, int start,
179       int before, int count)
180    {
181       // преобразование текста billEditText в double
182       try
183       {
184          currentBillTotal = Double.parseDouble(s.toString());
185       }  // завершение блока try
186       catch (NumberFormatException e)
187       {
188          currentBillTotal = 0.0; // по умолчанию в случае исключения
189       }  // завершение блока catch
190
191       // обновление EditTexts, содержащих стандартные
          // и пользовательские чаевые
192       updateStandard(); // обновление 10, 15 и 20 % EditTexts
193       updateCustom();   // обновление EditTexts
                            // с пользовательскими чаевыми
194    }  // конец метода onTextChanged
195
196    @Override
197    public void afterTextChanged(Editable s)
198    {
199    }  // конец метода afterTextChanged
200
201    @Override
202    public void beforeTextChanged(CharSequence s, int start, int count,
203       int after)
204    {
205    }  // завершение метода beforeTextChanged
206    }; // завершение billEditTextWatcher
207 }     // завершение класса TipCalculator
```

Переопределение метода onTextChanged интерфейса TextWatcher

Метод onTextChanged (строки 177–194) вызывается в случае изменения текста, отображаемого компонентом billEditText. Этот метод принимает четыре параметра (строки 178–179). В рассматриваемом примере используется лишь параметр CharSequence s, который содержит копию текста billEditText. Другие параметры указывают на то, что текст, количество символов которого определено параметром count, начинающийся в позиции, указанной параметром start, заменил фрагмент прежнего текста, длина которого определяется параметром before.

В строке 184 выполняется преобразование текста, введенного пользователем в компонент billEditText, в тип double. В строке 192 вызывается метод updateStandard, который выполняет обновление компонентов EditText 10 %, 15 % и 20 %, включающих сумму чаевых и итоговую сумму по счету (включая сумму чаевых). В строке 193 вызывается метод updateCustom, который обновляет компоненты EditText, включающие пользовательские чаевые и итоговый счет. При этом используется процентная ставка пользовательских чаевых, которая берется из SeekBar.

Методы beforeTextChanged и afterTextChanged интерфейса billEditTextWatcherTextWatcher

В этой главе не используются методы интерфейса TextWatcher. Поэтому эти методы переопределены пустыми методами (строки 196–205) для соответствия условиям разработки интерфейса.

4.6. Резюме

В этой главе вы создали свое первое интерактивное приложение Android — Tip Calculator. Сначала были вкратце рассмотрены функции этого приложения, потом в целях тестирования вычислялось значение стандартных и пользовательских чаевых на основе введенного счета. Вы выполнили подробные пошаговые инструкции по созданию графического интерфейса пользователя приложения с помощью инструментов ADT Plugin и среды Eclipse (редактор Visual Layout Editor, окна Outline и Properties). В следующих главах будут рассмотрены новые возможности GUI, которым ранее не уделялось внимание. И «под занавес» вас ожидает подробный анализ кода класса Activity приложения Tip Calculator, который определяет действия, происходящие при запуске приложения, а также логику приложения.

При разработке графического интерфейса приложения используется компонент TableLayout, с помощью которого выполняется распределение компонентов GUI по строкам и столбцам. Вы узнали о том, что каждая ячейка компонента TableLayout может быть пустой или содержать один компонент. Одновременно с этим каждая ячейка может быть структурой, содержащей другие компоненты. Вы использовали компоненты TableRow для создания строк в макете и освоили методику определения количества столбцов с помощью TableRow, содержащих большинство компонентов. Также вы узнали о том, что высота каждой строки определяется высотой самого «длинного» компонента в строке, а ширина столбца — шириной самого «широкого» элемента столбца (если не выбрано растяжение столбцов). Компоненты TextView применялись для создания подписей GUI-компонентов, а EditText — для получения суммы итогового счета, введенной

пользователем. Нефокусируемые компоненты EditText использовались для отображения различных значений чаевых и итогов, вычисляемых после выбора разных процентных ставок, а с помощью компонента SeekBar вводились пользовательские процентные ставки чаевых. Большая часть XML-разметки графического интерфейса пользователя была сгенерирована инструментами модуля ADT Plugin, но допускалось и непосредственное редактирование XML-разметки для настройки свойств, которые недоступны в окне Properties.

В процессе разработки приложения из этой главы был использован целый ряд возможностей объектно-ориентированного программирования на Java, включая классы, анонимные внутренние классы, объекты, методы, интерфейсы и наследование. Мы познакомились с «раздуванием» графического интерфейса пользователя из исходного XML-файла в экранное представление, освоили класс Android Activity и частично изучили жизненный цикл Activity. Мы рассмотрели метод onCreate, выполняющий инициализацию приложения при его запуске, и метод onSaveInstanceState, сохраняющий состояние приложения при изменении конфигурации устройства. В листингах главы использовался метод findViewById из класса Activity, который вызывается из метода onCreate и применяется для получения ссылок на компоненты GUI, с которыми приложение взаимодействует на программном уровне. Для объекта billEditText был определен анонимный внутренний класс, реализующий интерфейс TextWatcher. В результате приложение получило возможность вычислять новые значения чаевых и итогов после изменения текста EditText пользователем. Для компонента customSeekBar был определен анонимный внутренний класс, который реализует интерфейс OnSeekBarChangeListener. В результате приложение получило возможность вычислять новую сумму пользовательских чаевых и итогового счета после изменения пользователем значения процентной ставки. Это изменение осуществляется перемещением ползунка компонента SeekBar.

В следующей главе мы создадим приложение Favorite Twitter Searches, а заодно познакомимся с коллекциями и рассмотрим программный способ создания графического интерфейса пользователя, позволяющий динамически добавлять и удалять компоненты интерфейса в ответ на действия пользователя.

5

Приложение Favorite Twitter® Searches

Настройки Shared Preferences, кнопки, вложенные структуры, интенты, диалоговые окна Alert Dialogs, «раздувание» XML-разметки и файла манифеста

В этой главе...

- Взаимодействие пользователей с приложением посредством кнопок.
- Использование компонента ScrollView для отображения объектов, не помещающихся на экране.
- Динамическое создание компонентов GUI при взаимодействии с пользователем путем «раздувания» XML-разметки.
- Хранение пар «ключ/значение» для данных, связанных с приложением, с помощью SharedPreferences.
- Изменение пар «ключ/значение» для данных, связанных с приложением, с помощью SharedPreferences.Editor.
- Создание диалоговых окон AlertDialogs с помощью объекта AlertDialog.Builder.
- Программное открытие веб-сайта в окне браузера с помощью интентов.
- Программное скрытие виртуальной клавиатуры.

5.1. Введение

Приложение Favorite Twitter Searches помогает представить избранные (возможно длинные) строки поиска Твиттера в виде простых для запоминания, выбираемых

пользователем коротких имен тегов. Это позволит пользователям следовать за твитами в своих любимых темах. Поисковые запросы Твиттера могут быть настроены с помощью поисковых операторов Твиттера (dev.twitter.com/docs/using-search). Но при этом следует иметь в виду, что более сложные запросы имеют большую длину, требуют больше времени и более чувствительны к ошибкам при вводе на мобильных устройствах. Избранные поисковые запросы пользователя сохраняются на устройстве и могут быть доступны при каждом запуске приложения. На рис. 5.1, *а* показано приложение с несколькими сохраненными поисковыми запросами. При этом допускается сохранение пользователем большого количества поисковых запросов и их прокрутка в алфавитном порядке. Поисковые запросы и соответствующие им теги вводятся в компоненты EditText, находящиеся в верхней части экрана. С помощью компонента Save Button (Кнопка сохранения) каждый поисковый запрос добавляется в список избранного. В результате касания кнопки поиска поисковый запрос отправляется Твиттеру, а результаты поиска отображаются на экране веб-браузера. На рис. 5.1, *б* показан результат касания кнопки Google Button (Кнопка Google) — поиск твитов в Google, указанных поиском Твиттера from:Google. Чтобы изменить настройки поиска, воспользуйтесь кнопками Edit Button (Кнопки изменения), которые находятся справа от каждой кнопки поиска. С помощью этих кнопок можно настроить результаты поиска таким образом, чтобы получить лучшие результаты (в случае предварительного сохранения поисковых запросов как избранных). После касания кнопки Clear Tags Button (Очистить теги), находящейся в нижней части экрана, удаляются все ранее сохраненные поисковые запросы, находящиеся в списке избранного. Потом нужно подтвердить удаление, выбрав соответствующий параметр в появившемся диалоговом окне.



а *б*

Рис. 5.1. Приложение Favorite Twitter Searches: *а* — приложение с несколькими сохраненными поисковыми запросами; *б* — результат касания кнопки Google Button

5.2. Тестирование приложения Favorite Twitter Searches

Открытие и выполнение приложения

Откройте интегрированную среду разработки Eclipse, импортируйте проект приложения Favorite Twitter Searches. Выполните следующие действия:

1. *Откройте диалоговое окно* Import. Чтобы открыть это диалоговое окно, выполните команды File ▸ Import… (Файл ▸ Импорт…).
2. *Импортируйте проект приложения* Favorite Twitter Searches. В диалоговом окне Import раскройте узел General (Общие) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область), потом щелкните на кнопке Next > (Далее >) для выполнения шага Import Projects (Импорт проектов). Установите флажок Select root directory (Выбор корневого каталога), потом щелкните на кнопке Browse… (Просмотр…). В диалоговом окне Browse For Folder (Просмотр папки) выберите папку FavoriteTwitterSearches, находящуюся в папке examples книги, и щелкните на кнопке OK. Щелкните на кнопке Finish (Готово) для импорта проекта в Eclipse. После этого проект появится в окне Package Explorer, находящемся в левой части окна Eclipse.
3. Запустите приложение Favorite Twitter Searches. Для этого в среде Eclipse правой кнопкой мыши щелкните на проекте FavoriteTwitterSearches, находящемся в окне Package Explorer, потом в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android). В результате будет запущено на выполнение приложение Favorite Twitter Searches в среде AVD, которая создана и настроена в соответствии с указаниями из раздела «Предварительные действия», находящимся во введении к книге. Окно выполняющегося приложения показано на рис. 5.2.

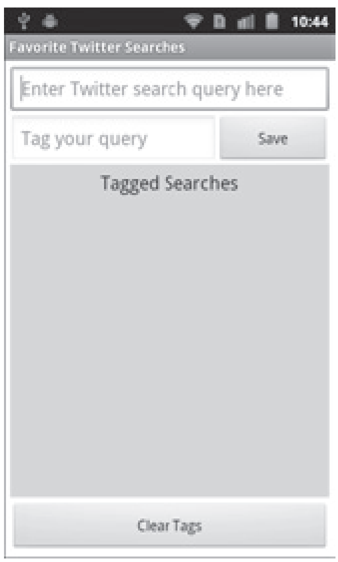
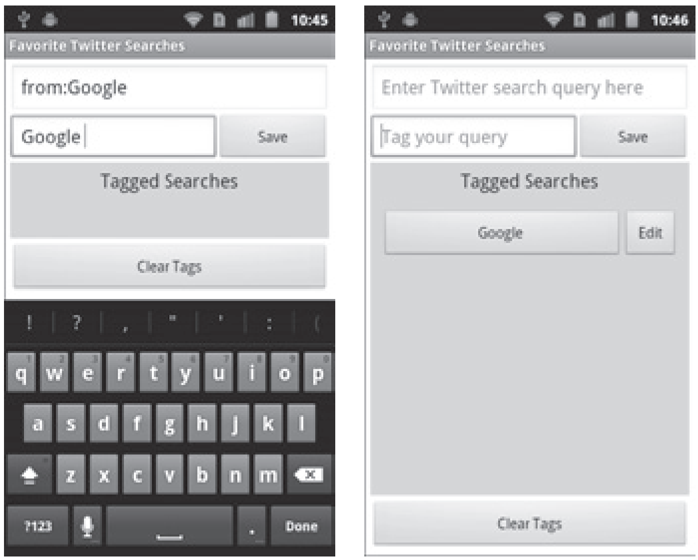


Рис. 5.2. Выполняющееся приложение Favorite Twitter Searches

Верхние два компонента EditText позволяют вводить новые поисковые запросы, а в разделе Tagged Searches (Тегированные поиски) отображаются ранее сохраненные поисковые запросы (в рассматриваемом примере подобные запросы отсутствуют).

Добавление нового избранного поиска

Введите фразу from:Google, задающую поисковый запрос, в верхний компонент EditText. В нижний компонент EditText введите Google (рис. 5.3, *а*). Введенное слово может использоваться в качестве краткого имени, отображающегося в разделе Tagged Searches. Чтобы сохранить поисковый запрос и скрыть экранную клавиатуру, нажмите кнопку Save Button. Под заголовком Tagged Searches появится кнопка Google Button (рис. 5.3, *б*). При этом также исчезнет экранная клавиатура. Это связанно с тем, что данное приложение скрывает экранную клавиатуру программным образом.



а *б*

Рис. 5.3. Ввод поискового запроса Твиттера: *а* — ввод поискового запроса Твиттера и тега поиска; *б* — окно приложения после сохранения поискового запроса и тега поиска

Изменение поискового запроса

Справа от каждой кнопки поиска находится кнопка редактирования (Edit Button). Коснитесь этой кнопки, и ваш запрос и тег поиска будут перезагружены в компоненты EditText, находящиеся в верхней части экрана приложения, и станут доступными для редактирования. В рассматриваемом примере ограничим поиск твитами, которые были созданы после 1 апреля 2011 года. Для этого в конец поискового запроса нужно добавить фразу since:2011-04-01 (рис. 5.4). Нажмите кнопку Save для обновления сохраненного поискового запроса.

ПРИМЕЧАНИЕ

Если изменить название тега, будет создана новая кнопка поиска. Эта возможность весьма полезна для создания нового запроса на основе ранее сохраненного.

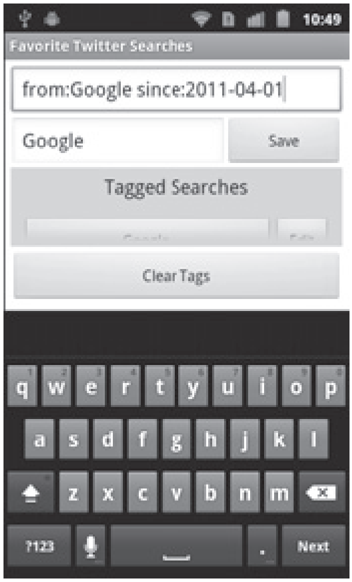


Рис. 5.4. Редактирование строки поиска в Твиттере

Просмотр результатов поиска в Твиттере

Чтобы просмотреть результаты поиска, нажмите кнопку Google search query Button. В результате откроется окно веб-браузера, и вы получите доступ к веб-сайту Твиттера и сможете получить и отобразить результаты поиска (рис. 5.5).

5.3. Обзор применяемых технологий

Приложение использует GUI-компоненты EditText, ScrollView и Button. Компонент ScrollView представляет собой объект ViewGroup, который может включать другие представления (Views), например макет (layout). Этот компонент обеспечивает *прокрутку* контента при его просмотре на экране. С помощью компонента ScrollView можно отображать на экране любые по величине списки сохраненных поисковых запросов, среди которых будут избранные запросы, которые могут помещаться на экране. Каждый поисковый запрос связан с компонентом Button — коснитесь этого компонента, чтобы начать поиск с помощью браузера.

SharedPreferences

В вашем распоряжении может быть один или несколько файлов, включающих пары «ключ/значение», которые связаны с каждым приложением. Эта особенность



Рис. 5.5. Просмотр результатов поиска

используется для манипулирования файлами, которые называются *поисками*. В этих файлах хранятся пары тегов и поисковые запросы Твиттера, созданные пользователем. Для считывания пар «ключ/значение» из файла используются объекты SharedPreferences (пакет android.content). Чтобы изменить содержимое файла, используются объекты SharedPreferences.Editor (пакет android.content). Ключи должны иметь строковый тип, а значения могут иметь строковый или примитивный тип.

С помощью метода refreshButtons осуществляется считывание данных в сохраненных поисках. Этот метод вызывается из метода onCreate, относящегося к классу Activity. Эта методика применяется, когда загруженный объем данных не слишком велик. После загрузки приложения Android создает основной поток, который называется *UI thread*. В этом потоке реализуются выполняемые GUI операции. Обратите внимание, что в потоке UI thread не рекомендуется выполнять интенсивные операции ввода/вывода, поскольку это может привести к резкому ухудшению отзывчивости приложения. Дополнительные сведения по этой теме приведены в главе 10.

Интенты

Обычно *интенты* используются для запуска деятельностей. Они указывают выполняемое *действие* и *данные*, по отношению к которым это действие будет выполнено. Как только пользователь коснется объекта Button, представляющего поиск, создается URL-ссылка, которая включает поисковый запрос Твиттера. С помощью создания нового интента (Intent) выполняется загрузка URL-ссылки в веб-браузер, потом интент передается методу startActivity, который класс Activity косвенным образом наследует из класса Context. Для просмотра URL-ссылки метод startActivity запускает веб-браузер устройства, в окне которого отображается содержимое. В данном приложении в качестве содержимого используются результаты поиска в Твиттере.

LayoutInflater

После ввода пользователем нового поискового запроса в пользовательский интерфейс добавляется новый ряд объектов Button. Один объект Button представляет поиск, а второй позволяет изменять поиск. Для программного создания этих компонентов GUI на основе заранее определенной XML-разметки используется компонент LayoutInflater. Компонент LayoutInflater «раздувает» файл XML-разметки, в результате чего создаются компоненты, которые были специфицированы в XML-коде. Затем определяется поисковый текст объекта Button, регистрируются обработчики событий каждого объекта Button и новые компоненты GUI связываются с пользовательским интерфейсом.

AlertDialog

Перед сохранением нового поискового запроса нужно, чтобы пользователь ввел сам запрос и тег. Если этого не произойдет и компонент EditText останется пустым, отобразится соответствующее сообщение для пользователя. Также требуется подтверждение пользователем удаления всех поисковых запросов после касания кнопки Clear Tags (Очистить теги). Необходимые сообщения и подтверждения отображаются с помощью объектов AlertDialog. Во время отображения этого диалогового окна (модальное диалоговое окно) блокируется взаимодействие пользователя с приложением. В следующих разделах будет показано, каким образом определять настройки для подобных диалоговых окон с помощью объекта AlertDialog.Builder, а затем с его помощью создавать AlertDialog.

Файл AndroidManifest.xml

Файл AndroidManifest.xml генерируется при создании приложения с помощью подключаемого модуля ADT Plugin в среде Eclipse. В этом файле определяются настройки, например название приложения, имя пакета, целевая SDK и минимальная версия SDK, имя объекта Activity для приложения и ряд других настроек. В конце главы вас ожидает обзор структуры этого файла. Также вы узнаете, каким образом в манифест добавить новую настройку, предотвращающую отображение виртуальной клавиатуры после первой загрузки приложения.

5.4. Создание графического интерфейса приложения и файлов ресурсов

В этом разделе будет создан графический интерфейс пользователя для приложения Favorite Twitter Searches. Вашему вниманию будет представлен код XML, сгенерированный модулем ADT Plugin для макета приложения. В первую очередь будут рассмотрены новые возможности и функции GUI, представлена завершенная XML-разметка и рассмотрены ключевые фрагменты XML-кода. Будет также создана вторая XML-разметка, динамическое «раздувание» которой создаст теги и кнопоки Edit для каждого поискового запроса. В результате приложение сможет загружать ранее сохраненные поисковые запросы и менять внешний вид в процессе добавления или удаления поисковых запросов пользователем.

5.4.1. Компонент main.xml TableLayout

Как и в главе 4, для создания основного макета приложения используется компонент TableLayout (рис. 5.6), который в данном случае включает пять строк и два столбца. Атрибуту android:stretchColumns компонента TableLayout присвоено значение "*". В результате все строки таблицы будут «растягиваемыми», то есть элементы в каждом столбце могут расширяться на весь экран.



Рис. 5.6. Строки и столбцы компонента TableLayout приложения Favorite Twitter Searches

На рис. 5.7 показаны названия компонентов GUI. При создании этих имен мы руководствовались соглашением о наименовании. Суть этого соглашения заключается в использовании имени класса GUI-компонента для каждого свойства Id компонента в XML-разметке и в каждом имени переменной в коде Java.

5.4.2. Создание проекта

Создайте новый проект Android и присвойте ему имя FavoriteTwitterSearches. В диалоговом окне New Android Project (Новый проект Android) укажите значения следующих параметров и нажмите кнопку Finish (Готово):

- Build Target (Операционная система): Android 2.3.3.
- Application name (Имя приложения): Favorite Twitter Searches.
- Package name (Название пакета): com.deitel.favoritetwittersearches.
- Create Activity (Создать деятельность): FavoriteTwitterSearches.
- Min SDK Version (Минимальная версия SDK): 10.



Рис. 5.7. Компоненты GUI приложения Favorite Twitter Searches именуются с помощью значений свойства Id

ПРИМЕЧАНИЕ

Эта версия SDK соответствует версии Android 2.3.3, хотя в этом приложении не будут использоваться функции, присущие именно Android 2.3.3. Если планируется выполнять это приложение на виртуальном устройстве Android (AVD) либо на реальном устройстве с более ранней версией Android, параметру Min SDK присвойте меньшее значение. Например, если выбрано значение 8 для этого параметра, приложение будет выполняться на версии Android 2.2 или более старшей.

5.4.3. Создание файлов ресурсов

В этом приложении литеральное значение цвета и несколько литеральных значений размеров хранятся в файлах colors.xml и dimen.xml соответственно. Подобные имена файлов используются по соглашению, а сами файлы находятся в папке приложения res/values. Цвет и размеры, хранящиеся в этих файлах, будут представлены в автоматически сгенерированном файле R.java с помощью константы, которую можно использовать в качестве ссылки на определенное значение. Чтобы создать каждый из перечисленных выше файлов, выполните следующие действия:

1. Щелкните правой кнопкой мыши на имени проекта в окне Package Explorer и выполните команды New ▸ Other... (Создать ▸ Другой...), затем в диалоговом окне New (Создать) в узле Android выберите параметр Android XML File (XML-файл Android). На экране появится диалоговое окно New Android XML File (Создать XML-файл Android).
2. В текстовое поле File (Файл) введите имя colors.xml.
3. В разделе What type of resource would you like to create? (Укажите тип создаваемого ресурса) выберите переключатель Values (Значения). После этого в папке проекта res/values появится новый файл.

4. Для завершения создания файла щелкните на кнопке Finish (Готово).
5. Повторите описанный выше процесс для создания файла dimen.xml.

Содержимое этих двух файлов показано в листингах 5.1–5.2. В XML-разметке проекта из этой главы используются значения цвета и размера, заданные в этих файлах. Также используются несколько предварительно определенных цветов Android из класса R.color. Как и в предыдущих приложениях, в файле strings.xml будут заданы многочисленные строковые ресурсы.

Файл colors.xml

XML-документы, представляющие ресурсы, должны включать элемент resources, с помощью которого определяются ресурсы. Внутри этого элемента (см. листинг 5.1) определяется единственное значение цвета, используемое в приложении (light_orange). Элемент color (строка 3) определяет атрибут name, который используется для ссылки на цвет, и шестнадцатеричное значение, определяющее цвет.

Листинг 5.1. Цвета, определенные в файле colors.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <resources>
3     <color name="light_orange">#8f90</color>
4  </resources>
```

Файл dimen.xml

В листинге 5.2 определяются элементы dimen, представляющие значения ширины тега поиска и кнопок Edit. Благодаря определению размеров в форме ресурсов можно использовать пиксельные значения, независимые от плотности (dp или dip), и пиксельные значения, независимые от масштаба (sp). В результате Android выполняет автоматическое преобразование в соответствующие пиксельные значения для выбранного устройства. В коде используются лишь фиксированные пиксельные размеры, в результате чего у пользователя появляется возможность вычисления подходящих пиксельных значений для каждого устройства вручную.

Листинг 5.2. Размеры, определенные в файле dimen.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
   <dimen name="tagButtonWidth">230dp</dimen>
   <dimen name="editButtonWidth">50dp</dimen>
</resources>
```

Файл strings.xml

В листинге 5.3 определяются строковые литеральные значения, используемые в приложении. В строке 4 определяется значение searchURL. Поисковые запросы пользователя добавляются к этой URL-ссылке перед отображением поиска Твиттера на экране веб-браузера устройства.

Листинг 5.3. Строки, определенные в файле strings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
   <string name="app_name">Favorite Twitter Searches</string>
   <string name="searchURL">http://search.twitter.com/search?q=</string>
   <string name="tagPrompt">Назначьте тег запросу</string>
   <string name="queryPrompt">Введите поисковый запрос Твиттера</string>
   <string name="taggedSearches">Тегированные поиски</string>
   <string name="edit">Изменить</string>
   <string name="clearTags">Очистить теги</string>
   <string name="save">Сохранить</string>
   <string name="erase">Удалить</string>
   <string name="cancel">Отмена</string>
   <string name="OK">OK</string>
   <string name="missingTitle">Текст отсутствует</string>
   <string name="missingMessage">
      Введите текст запроса и назначьте ему тег.</string>
   <string name="confirmTitle">Вы уверены?</string>
   <string name="confirmMessage">
      Сейчас будут удалены все сохраненные поисковые запросы</string>
</resources>
```

5.4.4. Добавление класса TableLayout и компонентов

С помощью технологий, описанных в главе 4, будет создан графический интерфейс пользователя (см. рис. 5.6–5.7). Начнем с создания базовой структуры и элементов управления, потом настроим свойства элементов управления для завершения проекта. После добавления компонентов в каждую строку TableLayout установим значения свойств Id и Text этих компонентов (см. рис. 5.7). В процессе построения GUI поместите литеральные строковые значения в файл strings.xml, находящийся в папке res/values приложения. С помощью окна Outline добавьте компоненты в требуемые TableRows макета TableLayout.

Шаг 1. Удаление и повторное создание файла main.xml

Для создаваемого в этой главе приложения вместо стандартного файла main.xml воспользуйтесь другим подобным файлом с объектом TableLayout, позволяющим реализовать относительное расположение компонентов. Чтобы заменить стандартный файл main.xml, выполните следующие действия:

1. Чтобы удалить файл mail.xml, щелкните на нем (в папке проектов /res/layout) и в контекстном меню выберите параметр Delete (Удалить).
2. Правой кнопкой мыши щелкните на папке layout и выберите команды New ▸ Other... (Создать ▸ Другое...). Отобразится диалоговое окно New (Создать).
3. В узле Android выберите параметр Android XML File (XML-файл Android). После щелчка на кнопке Next> (Далее>) появится диалоговое окно New Android XML File (Создать XML-файл Android).
4. Выберите название файла main.xml и выделите TableLayout, потом щелкните на кнопке Finish.

Шаг 2. Конфигурирование Visual Layout Editor для использования подходящей библиотеки Android SDK

В раскрывающемся списке селектора SDK, находящемся в правой верхней части вкладки Graphical Layout, выберите параметр Android 2.3.3 (как на рис. 3.7). В результате разрабатываемый графический интерфейс будет совместим с устройствами Android 2.3.3.

Шаг 3. Настройка размеров и разрешения окна Visual Layout Editor

В раскрывающемся списке Device Configurations (Конфигурации устройств), находящемся в левой верхней части вкладки Graphical Layout (см. рис. 3.11), выберите параметр 3.7in WVGA (Nexus One). В результате будет настроена область конфигурирования для устройств с экраном, имеющим разрешение 480×800 (WVGA).

Шаг 4. Конфигурирование компонента TableLayout

В окне Outline выберите параметр TableLayout и установите следующие значения:

- Background (Фон): @android:color/white;
- Id (Идентификатор): @+id/tableLayout;
- Padding (Отступ): 5dp;
- Stretch columns (Растяжение столбцов): *.

Цвет Background определяется с помощью одного из заранее заданных цветовых значений Android, white (белый), находящихся в класса R.color. Названия цветов, определенных в этом классе, можно найти на веб-сайте developer.android.com/reference/android/R.color.html.

Чтобы получить доступ к заранее определенному ресурсу цвета, воспользуйтесь инструкцией @android:color/, а потом укажите название ресурса.

По умолчанию макет занимает весь экран, поскольку свойствам Layout width и Layout height присвоено значение match_parent. Параметру Padding присваивается значение 5dp, в результате чего вдоль границы всего макета создается кайма, шириной в 5 независимых от плотности пикселей. Значение, присвоенное параметру Stretch, определяет растяжение столбцов по горизонтали на всю ширину макета.

Шаг 5. Добавление компонентов TableRow

С помощью окна Outline добавьте в TableLayout пять компонентов TableRow (см. главу 4). Перед добавлением очередного TableRow выделяйте TableLayout, чтобы обеспечить корректное вложение TableRows в TableLayout. Измените свойства Id для пяти компонентов TableRow, выбрав значения tableRow0, tableRow1, tableRow2, tableRow3 и tableRow4 соответственно. Также выделите каждый компонент TableRow и присвойте его свойству Layout width значение match_parent. В результате строки растянутся на всю ширину макета.

Шаг 6. Добавление компонентов в TableRows

Используя рис. 5.6 и 5.7 в качестве руководства, добавьте в макет EditText, Button, TextView и ScrollView. Также поместите элемент TableLayout внутри компонента ScrollView. Присвойте элементам имена (см. рис. 5.7). Просмотрите XML-элементы в файле main.xml (листинг 5.4), чтобы получить представление о значениях, присвоенных атрибутам

для каждого GUI-компонента. Новые и ключевые функции, определенные в листинге 5.4, будут рассмотрены позднее.

Листинг 5.4. XML-разметка приложения Favorite Twitter Search

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android=http://schemas.android.com/apk/res/android
   android:id="@+id/tableLayout" android:layout_width="match_parent"
   android:layout_height="match_parent" android:padding="5dp"
   android:stretchColumns="*" android:background="@android:color/white">

   <!-- tableRow0 -->
   <TableRow android:id="@+id/tableRow0"
     android:layout_height="wrap_content"
      android:layout_width="match_parent">
      <EditText android:layout_width="match_parent"
         android:layout_height="wrap_content" android:layout_span="2"
         android:inputType="text" android:id="@+id/queryEditText"
         android:hint="@string/queryPrompt"
         android:imeOptions="actionNext">
      </EditText>
   </TableRow>

   <!-- tableRow1 -->
   <TableRow android:id="@+id/tableRow1"
      android:layout_height="wrap_content"
      android:layout_width="match_parent">
      <EditText android:layout_height="wrap_content"
         android:hint="@string/tagPrompt" android:inputType="text"
         android:id="@+id/tagEditText" android:imeOptions="actionDone"
         android:layout_gravity="center_vertical"></EditText>
      <Button android:id="@+id/saveButton"
         android:layout_height="wrap_content"
         android:layout_width="wrap_content"
         android:layout_gravity="center_vertical"
         android:text="@string/save"></Button>
      </TableRow>

      <!-- tableRow2 -->
      <TableRow android:id="@+id/tableRow2"
         android:layout_height="wrap_content"
         android:layout_width="match_parent"
         android:background="@color/light_orange">

         <TextView android:layout_height="wrap_content"
            android:id="@+id/taggedSearchesTextView"
            android:text="@string/taggedSearches"
            android:layout_width="match_parent"
            android:layout_gravity="center_horizontal"
            android:layout_span="2" android:textSize="18sp"
            android:textColor="@android:color/black"
```

продолжение ⇨

Листинг 5.4 *(продолжение)*

```
47              android:padding="5dp"></TextView>
48           </TableRow>
49
50           <!-- tableRow3 -->
51           <TableRow android:id="@+id/tableRow3"
52              android:background="@color/light_orange"
53              android:layout_height="wrap_content"
54             android:layout_width="match_parent"> android:layout_weight="1"
55
56             <ScrollView android:id="@+id/queryScrollView"
57                android:layout_width="match_parent"
58                android:layout_span="2" android:padding="5dp">
59                <TableLayout android:id="@+id/queryTableLayout"
60                   android:layout_width="match_parent"
61                 android:layout_height="match_parent" android:padding="5dp"
62                   android:stretchColumns="*"> </TableLayout>
63              </ScrollView>
64           </TableRow>
65
66           <!-- tableRow4 -->
67           <TableRow android:id="@+id/tableRow4"
68              android:layout_height="wrap_content"
69              android:layout_width="match_parent">
70
71              <Button android:layout_width="wrap_content"
72                 android:layout_height="wrap_content"
73                 android:text="@string/clearTags"
74                 android:id="@+id/clearTagsButton"
75                 android:layout_span="2"
                      android:layout_marginTop="5dp"></Button>
76           </TableRow>
77        </TableLayout>
```

Ключевые компоненты файла main.xml

В главе 4 значение атрибута android:layout_span (строки 12, 45, 58 и 75) устанавливалось непосредственно в XML-коде, поскольку этот атрибут не отображается в окне Properties, в котором выбран режим проектирования. Ресурсы, находящиеся в файлах colors.xml, dimen.xml и strings.xml, которые используются для настройки различных свойств компонентов GUI, будут описаны в следующем списке. Чтобы получить доступ к различным значениям ресурсов в XML-коде, выполните следующие действия:

- *Строки*. Укажите инструкции @string/, которые следуют за именем ресурса. Например, в строках 14 и 31 заданы значения строкового ресурса для атрибута android:hint, соответствующего компонентам EditText. Этот атрибут отображает внутри элемента EditText подсказку, которая поможет пользователю осознать предназначение компонента EditText. Другие строковые ресурсы используются для представления текста в различных GUI-компонентах, например Button (строки 31 и 73) и TextView (строка 41).

- *Цвета*. Воспользуйтесь инструкцией @color/, которая следует после названия ресурса. Например, в строках 38 и 52 определяется ресурс цвета, определяющий фоновый цвет для компонентов tableRow2 и ScrollView соответственно.

В строках 15 и 25 используется атрибут android:imeOptions компонента EditText, с помощью которого настраиваются параметры для текущего метода ввода. Например, если элемент queryEditText получает фокус и отображается виртуальная клавиатура, эта клавиатура содержит кнопку Next (Далее). Эта кнопка определяется путем присваивания атрибуту android:imeOptions значения actionNext (строка 15). Если пользователь касается этой кнопки, фокус перемещается к следующему компоненту, который принимает пользовательский ввод, — tagEditText. Как только компонент tagEditText получает фокус, на виртуальной клавиатуре появляется кнопка Done (Готово). Эта кнопка определяется путем присваивания атрибуту android:imeOptions значения actionDone (строка 25). Как только пользователь касается этой кнопки, система скрывает виртуальную клавиатуру.

В строках 27–31 и 71–75 определяются компоненты Button, применяемые для сохранения и очистки всех ранее сохраненных поисковых запросов соответственно. В строках 56–63 определяется компонент ScrollView, включающий TableLayout (строки 59–62), которые отображают кнопки поиска (причем отображение задается программно). Атрибуту android:stretchColumns компонента TableLayout присваивается значение "*", в результате чего содержимое каждого компонента TableRow, которое программным образом помещается в TableLayout, может растягиваться на всю ширину макета. Если имеется несколько кнопок поиска, отображаемых на экране, можно выполнить перетаскивание экрана пальцем вверх или вниз в области компонента ScrollView, чтобы выполнить прокрутку объектов Button в TableLayout. Как показано в разделе 5.5, компонент TableLayout будет включать TableRows, каждый из которых содержит кнопки поиска и редактирования.

В строке 54 атрибуту android:layout_weight компонента tableRow3 присваивается значение 1. В результате компоненту tableRow3 присваивается высший приоритет по сравнению с другими строками в процессе изменения размеров основной структуры таблицы, выполняемого при наличии доступного пространства. Поскольку атрибут android:layout_weight определяется лишь для компонента tableRow3, этот компонент растягивается по вертикали, занимая все свободное место, в котором отсутствуют другие строки.

5.4.5. Создание компонента TableRow, отображающего кнопки Search и Edit

На этом этапе будет определен компонент TableRow, который будет «раздут», чтобы программным образом создать каждую кнопку поиска и соответствующую ей кнопку редактирования. В разделе 5.5 рассматриваются методики, используемые для отображения кнопок путем конфигурирования Button и добавления элементов TableRow в компонент queryTableLayout (см. листинг 5.4, строки 59–62). Чтобы создать другой файл XML-разметки, выполните следующие действия:

1. Щелкните правой кнопкой мыши на папке layout и в контекстном меню выберите параметры New ▸ Other... для отображения диалогового окна New.
2. В узле Android выберите параметр Android XML File и щелкните на кнопке Next> для отображения диалогового окна New Android XML File.

3. В текстовое поле File (Файл) введите имя new_tag_view.xml.
4. В разделе What type of resource would you like to create? (Тип создаваемого ресурса) установите переключатель Layout (Макет). В результате новый файл new_tag_view.xml запишется в папку проекта res/layout.
5. В нижней части диалогового окна выберите *корневой элемент* для нового макета. Выберите компонент TableRow.
6. Щелкните на кнопке Finish для создания файла, который незамедлительно откроется в представлении XML.
7. Выберите вкладку Graphical Layout (Графический макет) в окне редактора Visual Layout Editor, в раскрывающемся списке SDK Selector, находящемся в правой части вкладки Graphical Layout, выберите параметр Android 2.3.3. В раскрывающемся списке Device Configurations (Конфигурации устройства), находящемся в верхней левой части вкладки Graphical Layout, выберите параметр 3.7in WVGA (Nexus One).

Добавьте в макет два объекта Button. Сконфигурируйте свойства Button и layout (см. листинг 5.5). Атрибуту android:text значение newTagButton не присваивается, поскольку это значение присваивается определенному тегу поиска во время программного создания объектов Button. Атрибуту android:background компонента TableLayout присваивается заранее определенный цвет transparent (прозрачный), как указано в строке 6. В результате появится возможность «смотреть сквозь» фоновый цвет компонента ScrollView при связывании элемента TableRow с ScrollView. По умолчанию компонент ScrollView имеет тот же фоновый цвет, что и родительский элемент (в данном случае tableRow3). В строках 9 и 12 конструкция @dimen/ следует после имени ресурса размеров, определяя ширину объектов Button.

Листинг 5.5. Компонент newTagTableRow, который «раздувается» программным образом

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <TableRow xmlns:android=http://schemas.android.com/apk/res/android
 3    android:id="@+id/newTagTableRow"
 4    android:layout_width="match_parent"
 5    android:layout_height="wrap_content"
 6    android:background="@android:color/transparent">
 7
 8    <Button android:id="@+id/newTagButton"
 9      android:layout_width="@dimen/tagButtonWidth"
10       android:layout_height="wrap_content"></Button>
11    <Button android:id="@+id/newEditButton"
12       android:layout_width="@dimen/editButtonWidth"
13       android:layout_height="wrap_content"
14       android:text="@string/edit"></Button>
15  </TableRow>
```

5.5. Создание приложения

В листингах 5.6–5.16 представлен код приложения Favorite Twitter Searches, реализованного в единственном классе FavoriteTwitterSearches, который расширяет класс Activity.

Операторы package и import

В листинге 5.6 приведены операторы package и import, используемые в приложения. С помощью оператора package (строка 4) определяется, что класс, находящийся в этом файле, является частью пакета i com.deitel.favoritetwittersearches. Эта строка включается автоматически средой разработки (IDE) при создании проекта. Операторы import (строки 6–23) импортируют различные классы и интерфейсы, используемые приложением.

Листинг 5.6. Операторы package и import, используемые в приложении FavoriteTwitterSearches

```
 1 // FavoriteTwitterSearches.java
 2 // Хранит поисковые запросы Твиттера и теги, облегчающие их открытие
 3 // в окне браузера.
 4 package com.deitel.favoritetwittersearches;
 5
 6 import java.util.Arrays;
 7
 8 import android.app.Activity;
 9 import android.app.AlertDialog;
10 import android.content.Context;
11 import android.content.DialogInterface;
12 import android.content.Intent;
13 import android.content.SharedPreferences;
14 import android.net.Uri;
15 import android.os.Bundle;
16 import android.view.LayoutInflater;
17 import android.view.View;
18 import android.view.View.OnClickListener;
19 import android.view.inputmethod.InputMethodManager;
20 import android.widget.Button;
21 import android.widget.EditText;
22 import android.widget.TableLayout;
23 import android.widget.TableRow;
24
```

В строке 6 импортируется класс Arrays из пакета java.util. С помощью метода sort этого класса будут сортироваться теги, представляющие каждый поиск. В результате сохраненные поисковые запросы отображаются в алфавитном порядке. При этом используются операторы import, относящиеся к классам, рассматриваемым в этой главе.

- Класс AlertDialog из пакета android.app (строка 9) применяется для отображения диалоговых окон.
- Класс Context из пакета android.content (строка 10) обеспечивает доступ к информации о среде, в которой выполняется приложение, а также обеспечивает доступ к различным службам Android. Константа из этого класса вместе с объектом LayoutInflater (рассматривается позднее) для упрощения динамической загрузки новых компонентов GUI.

- Класс DialogInterface из пакета android.content (строка 11) включает вложенный интерфейс OnClickListener. С помощью этого интерфейса обрабатываются события, которые имеют место после касания пользователем кнопки AlertDialog.
- Класс Intent из пакета android.content (строка 12) обеспечивает возможность работы с интентами (Intents). С помощью класса Intent определяется выполняемое *действие* и обрабатываемые *данные* — в Android объекты Intent применяются для запуска соответствующих деятельностей.
- Класс SharedPreferences из пакета android.content (строка 13) применяется для оперирования устойчивыми парами «ключ/значение», которые сохранены в файлах, связанных с приложением.
- Класс Uri из пакета android.net (строка 14) позволяет преобразовывать URL-ссылки из Интернета в формат, требуемый классу Intent для запуска веб-браузера устройства. Дополнительные сведения относительно URI- и URL-ссылок приведены в разделе 5.5.
- Класс LayoutInflater из пакета android.view (строка 16) позволяет динамически «раздувать» файл XML-разметки, чтобы создавать GUI-компоненты макета.
- Класс InputMethodManager из пакета android.view.inputmethod (строка 19) обеспечивает скрытие виртуальной клавиатуры после сохранения поискового запроса пользователем.
- Пакет android.widget (строки 20–23) включает виджеты (компоненты GUI) и макеты, которые используются в Android GUI. Класс Button из пакета android.widget (line 20) представляет простую кнопку, после нажатия которой выполняется определенное действие. Интерфейс View.OnClickListener из пакета android.view (строка 18) используется для определения кода, вызываемого после касания пользователем компонента Button.

Класс Activity для приложения Favorite Twitter Searches

Класс FavoriteTwitterSearches (см. листинги 5.7–5.16) — это единственный класс Activity для приложения Favorite Twitter Searches. После создания проекта FavoriteTwitterSearches модуль ADT Plugin генерирует этот класс в виде подкласса класса Activity (см. листинг 5.7, строка 26) и поддерживает оболочку для переопределенного метода onCreate. Этот метод *должен* быть переопределен для каждого подкласса из класса Activity.

Листинг 5.7. Класс FavoriteTwitterSearches является подклассом класса Activity

```
25  // главный (и единственный) класс Activity приложения
       // Favorite Twitter Searches
26  public class FavoriteTwitterSearches extends Activity
27 {
28    private SharedPreferences savedSearches; // избранные
                            // поисковые запросы пользователя
29    private TableLayout queryTableLayout; // кнопки поиска
30    private EditText queryEditText; // ввод запросов пользователя
31    private EditText tagEditText;   // ввод тега поискового запроса
32
```

В строке 28 объявляется экземпляр переменной savedSearches из объектов SharedPreferences. В объектах SharedPreferences хранятся *пары «ключ/значение»*, причем ключи имеют строковый тип данных, а значения могут иметь примитивный или строковый тип данных. Объект SharedPreferences применяется для хранения сбереженных пользователем поисковых запросов. В строке 29 объявляется объект TableLayout, используемый для получения доступа к части GUI, в которой программным методом отображаются новые кнопки. В строках 30–31 объявляются два компонента EditText, используемые для получения доступа к запросам и тегам, введенным пользователем в верхнюю часть окна приложения.

Переопределенный метод OnCreate из класса Activity

Метод onCreate (см. листинг 5.8) вызывается системой в следующих случаях:

- при загрузке приложения;
- если процесс приложения был «убит» операционной системой, пока приложение находилось в фоновом режиме, а затем выполнение приложения было возобновлено;
- при каждом изменении конфигурации, например при вращении устройства пользователем либо при выдвижении/скрытии физической клавиатуры.

Этот метод инициализирует экземпляры переменных из класса Activity и GUI-компоненты. Причем он упрощен до предела в целях ускорения загрузки приложений. В строке 37 выполняется необходимый вызов для метода onCreate суперкласса. Как и в случае с предыдущим приложением, в результате вызова метода setContentView (строка 38) передается константа R.layout.main для «раздувания» GUI на основе файла main.xml. Метод setContentView использует эту константу для загрузки соответствующего XML-документа, затем выполняет «раздувание» GUI.

Листинг 5.8. Переопределение метода onCreate из класса Activity

```
33      // вызывается при первом создании деятельности
34      @Override
35      public void onCreate(Bundle savedInstanceState)
36      {
37         super.onCreate(savedInstanceState); // вызов версии суперкласса
38         setContentView(R.layout.main);      // настройка макета
39
40        // объект SharedPreferences, который содержит сохраненные
          //поисковые запросы пользователя
41         savedSearches = getSharedPreferences("searches", MODE_PRIVATE);
42
43       // получение ссылки на queryTableLayout
44       queryTableLayout =
45          (TableLayout) findViewById(R.id.queryTableLayout);
46
47       // получение ссылок на два компонента EditTexts и кнопку Save
48       queryEditText = (EditText) findViewById(R.id.queryEditText);
49       tagEditText = (EditText) findViewById(R.id.tagEditText);
50
```

продолжение ⇨

Листинг 5.8 *(продолжение)*

```
51        // регистрация слушателей для кнопок Save и Clear Tags
52        Button saveButton = (Button) findViewById(R.id.saveButton);
53        saveButton.setOnClickListener(saveButtonListener);
54        Button clearTagsButton =
55           (Button) findViewById(R.id.clearTagsButton);
56        clearTagsButton.setOnClickListener(clearTagsButtonListener);
57
58        refreshButtons(null); // добавление ранее сохраненных
                              // поисковых запросов в GUI
59    }   // конец метода onCreate
60
```

В строке 41 используется метод getSharedPreferences (косвенно унаследован из класса Context) для получения объекта SharedPreferences, который может считывать *пары «тег/запрос»*, сохраненные ранее в файле «поисков». Первый аргумент определяет имя файла, содержащего данные. Второй аргумент определяет доступность файла и может принимать одно из следующих значений:

- **MODE_PRIVATE.** Файл доступен *только* для данного приложения. В большинстве случаев эту константу можно использовать в качестве второго аргумента для getSharedPreferences.
- **MODE_WORLD_READABLE.** Любое приложение, выполняющееся на устройстве, может выполнять *считывание* из файла.
- **MODE_WORLD_WRITABLE.** Любое приложение, выполняющееся на устройстве, может записывать в файл.

Эти константы могут комбинироваться вместе с поразрядным оператором OR (|).

Поскольку приложение не выполняет считывание больших объемов данных, метод onCreate выполняет достаточно быструю загрузку поисковых запросов. Никогда не реализуйте длинные запросы данных в потоке UI thread, если не хотите столкнуться с диалоговым окном Application Not Responding (ANR), которое обычно появляется после пяти секунд «простоя» системы. Дополнительные сведения о диалоговых окнах ANR и создании «отзывчивых» приложений можно найти в документе developer.android.com/guide/practices/design/responsiveness.html.

В строках 44–49 находятся ссылки на компоненты queryTableLayout, queryEditText и tagEditText для инициализации соответствующих экземпляров переменных. В строках 52–56 находятся ссылки на компоненты saveButton и clearTagsButton и выполняется регистрация соответствующих слушателей. И наконец, в строке 58 вызывается объект refreshButtons (см. листинг 5.9), который используется для создания объектов Button, предназначенных для ранее сохраненных поисков, и кнопок Edit, с помощью которых можно изменять поиски.

Метод refreshButtons из класса FavoriteTwitterSearches

Метод refreshButtons из класса FavoriteTwitterSearches (см. листинг 5.9) создает и отображает новый тег запроса и кнопки редактирования либо для только что сохраненного поиска (если аргумент не равен null), либо для всех поисковых запросов (если аргумент равен null).

Объекты Button отображаются в алфавитном порядке, что облегчает поиск необходимых кнопок пользователем. В строках 66–67 получаем строковый массив, представляющий ключи объекта SharedPreferences. Метод getAll объекта SharedPreferences возвращает карту (Map), которая включает все пары «ключ/значение». Затем вызывается метод keySet для данного объекта, с помощью которого создается объект Set для ключей. И напоследок вызывается метод toArray (с пустым строковым массивом в качестве аргумента) объекта Set, с помощью которого выполняется преобразование объекта Set в строковый массив, который затем сортируется в строке 68. Метод Arrays.sort (статический метод класса Arrays из пакета java.util) сортирует массив по первому аргументу. Поскольку при вводе тегов пользователем могут использоваться символы верхнего и нижнего регистров, в данном случае реализована *сортировка, независимая от регистра символов*. Эта сортировка выполняется путем передачи методу Arrays.sort заранее определенного объекта Comparator<String> String.CASE_INSENSITIVE_ORDER в качестве второго аргумента.

Листинг 5.9. Метод refreshButtons из класса FavoriteTwitterSearches повторно создает и отображает новые теги поиска и кнопки редактирования для всех сохраненных поисковых запросов

```
61  // пересоздание поискового тега и кнопки Edit
    // для всех сохраненных поисковых запросов;
62  // передача null для создания всех тегов и кнопок Edit.
63  private void refreshButtons(String newTag)
64  {
65     // store saved tags in the tags array
66     String[] tags =
67        savedSearches.getAll().keySet().toArray(new String[0]);
68   Arrays.sort(tags, String.CASE_INSENSITIVE_ORDER); // сортировка по тегу
69
70     // после добавления нового тега вставьте компонент GUI
       // в подходящее место
71     if (newTag != null)
72     {
73        makeTagGUI(newTag, Arrays.binarySearch(tags, newTag));
74     }  // конец блока if
75     else // отображение GUI для всех тегов
76     {
77        // отображение всех сохраненных поисковых запросов
78        for (int index = 0; index < tags.length; ++index)
79           makeTagGUI(tags[index], index);
80     }  // конец блока else
81   } // конец метода refreshButtons
82
```

В строках 71–80 определяется, будет ли метод вызываться для создания GUI, предназначенного для выполнения нового поиска либо для выполнения сохраненных поисков. В строке 73 вызывается метод makeTagGUI (см. листинг 5.11), который включает GUI для нового тега. Вызов Arrays.binarySearch во втором аргументе ищет точку вставки, чтобы поддерживать отсортированные по алфавиту кнопки тегов. Если

метод refreshButtons вызывается с аргументом null, в строках 78–79 вызывается метод makeTagGUI для каждого сохраненного поиска.

Метод makeTag из класса FavoriteTwitterSearches

Метод makeTag из класса FavoriteTwitterSearches (см. листинг 5.10) добавляет новый поиск в savedSearches либо изменяет существующий поиск. В строке 87 используется метод getString из класса SharedPreferences для поиска предыдущего значения (при его наличии), связанного с тегом. Если тег отсутствует в файле, возвращается второй аргумент (в данном случае null). В рассматриваемом случае метод также вызывает метод refreshButtons (строка 96), чтобы добавить GUI для нового поиска.

Листинг 5.10. Метод makeTag из класса FavoriteTwitterSearches добавляет новый поиск в файл сохранения, потом переустанавливает Button

```
83  // добавление нового поиска в файл, переустановка всех Button
84  private void makeTag(String query, String tag)
85  {
86     // originalQuery может быть null при изменении существующего запроса
87     String originalQuery = savedSearches.getString(tag, null);
88
89     // получение SharedPreferences.Editor для хранения новой
       // пары "тег/запрос"
90     SharedPreferences.Editor preferencesEditor = savedSearches.edit();
91     preferencesEditor.putString(tag, query); //сохранение текущего поиска
92     preferencesEditor.apply(); // сохранение обновленных настроек
93
94        // Если новый запрос, добавляется соответствующий GUI
95        if (originalQuery == null)
96           refreshButtons(tag); // добавляется новая кнопка для тега
97        }  // конец метода makeTag
98
```

В строках 90–92 добавляется новый тег или изменяется соответствующее значение существующего тега. Чтобы изменить файл, связанный с объектом SharedPreferences, сначала нужно вызвать метод edit, с помощью которого получить доступ к объекту SharedPreferences.Editor (строка 90). Этот объект поддерживает методы, выполняющие добавление пар «ключ/значение», удаление пар «ключ/значение» и изменение значения, связанного с определенным ключом в файле SharedPreferences. В строке 91 вызывается метод puttring этого объекта, с помощью которого сохраняется новый тег поиска (ключ) и запрашивается соответствующее значение. В строке 92 подтверждаются изменения в файле «поисков» путем вызова метода apply объекта SharedPreferences.Editor, который вносит изменения в файл.

Метод makeTagGUI из класса FavoriteTwitterSearches

Метод makeTagGUI из класса FavoriteTwitterSearches (см. листинг 5.11) добавляет в компонент queryTableLayout одну новую строку, содержащую тег, и кнопку Edit. Чтобы выполнить эту операцию, сначала «раздуйте» макет new_tag_view.xml, который был создан в разделе 5.4.5. Этот макет включает компонент TableRow, а также newTagButton и newEditButton.

В Android поддерживается служба, с помощью которой выполняется «раздувание» макета. Чтобы воспользоваться этой службой, следует получить ссылку на нее (строки 103–104) путем вызова метода getSystemService, унаследованного из класса Activity. При вызове этого метода используется аргумент Context.LAYOUT_INFLATER_SERVICE. Метод getSystemService может возвращать ссылки на различные системные службы, но от вас потребуется привести результат к типу LayoutInflater. В строке 107 вызывается метод «раздувания» из класса LayoutInflater с константой R.layout.new_tag_view, которая представляет макет new_tag_view.xml. При этом возвращается ссылка представлению (View), которое фактически представляет собой компонент TableRow, включающий Button. В строках 110–113 происходит получение ссылки на элемент newTagButton, настраивается его текст в соответствии со значением тега и выполняется регистрация его OnClickListener. В строках 116–118 производится получение ссылки на newEditButton и выполняется регистрация соответствующих OnClickListener. В строке 121 добавляется newTagView в queryTableLayout в месте, на которое указывает индекс.

Листинг 5.11. Метод makeTagGUI из класса FavoriteTwitterSearches создает тег и кнопки Edit Button для одного поиска, а затем добавляет их в queryTableLayout, в позицию

```
 99 // добавление в GUI кнопки нового тега и кнопок редактирования
100 private void makeTagGUI(String tag, int index)
101 {
102    // получение ссылки на службу LayoutInflater
103    LayoutInflater inflater = (LayoutInflater) getSystemService(
104       Context.LAYOUT_INFLATER_SERVICE);
105
106    // «раздувание» new_tag_view.xml для создания кнопок
       // нового тега и редактирования
107    View newTagView = inflater.inflate(R.layout.new_tag_view, null);
108
109    // получение newTagButton, настройка текста и регистрация слушателя
110    Button newTagButton =
111       (Button) newTagView.findViewById(R.id.newTagButton);
112    newTagButton.setText(tag);
113    newTagButton.setOnClickListener(queryButtonListener);
114
115    // получение newEditButton и регистрация его слушателя
116    Button newEditButton =
117       (Button) newTagView.findViewById(R.id.newEditButton);
118    newEditButton.setOnClickListener(editButtonListener);
119
120    // получение кнопок нового тега и редактирования
       // для queryTableLayout
121    queryTableLayout.addView(newTagView, index);
122 } // конец makeTagGUI
123
```

Метод clearButtons из класса FavoriteTwitterSearches

Метод clearButtons (см. листинг 5.12) удаляет все кнопки (Button) сохраненных поисковых запросов из приложения. В строке 128 вызывается метод removeAllViews класса queryTableLayout, который удаляет все вложенные компоненты TableRow, включающие объекты Button.

Листинг 5.12. Метод clearButtons из класса FavoriteTwitterSearches удаляет из приложения все объекты Button, представляющие сохраненные поисковые запросы

```
124    // удаляет кнопки всех сохраненных поисков из приложения
125    private void clearButtons()
126    {
127       // удаление всех кнопок сохраненных поисков
128       queryTableLayout.removeAllViews();
129    }  // конец метода clearButtons
130
```

Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события saveButton

В строках 132–170 (см. листинг 5.13) создается анонимный внутренний класс объекта saveButtonListener, который реализует интерфейс OnClickListener. В строке 53 зарегистрирован слушатель saveButtonListener в качестве обработчика событий объекта saveButtons. В строках 134–169 реализован метод OnClick интерфейса OnClickListener. Если пользователь введет запрос и тег (строки 138–139), метод вызовет метод makeTag (листинг 5.10), который сохраняет пару «тег/запрос» (строки 141–142), очищает оба компонента EditText (строки 143–144) и скрывает виртуальную клавиатуру (строки 147–149).

Если пользователь одновременно не ввел запрос и тег, метод отображает диалоговое окно AlertDialog (строки 151–168), в котором пользователю сообщается о том, что ему нужно ввести одновременно запрос и тег. Воспользуйтесь объектом AlertDialog.Builder (создан в строках 154–155) для конфигурирования и создания объекта AlertDialog. Аргумент конструктора — объект Context, в котором отображается диалоговое окно. В рассматриваемом случае это класс FavoriteTwitterSearches Activity, доступ к которому открывается по ссылке. Поскольку доступ открывается из анонимного внутреннего класса, следует полностью квалифицировать его, указав имя класса. В строке 157 определяется заголовок AlertDialog с помощью строкового ресурса R.string.missingTitle. Заголовок отображается в верхней части диалогового окна.

Листинг 5.13. Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события saveButton

```
131 // создание нового объекта Button и добавление в ScrollView
132 public OnClickListener saveButtonListener = new OnClickListener()
133 {
134    @Override
135    public void onClick(View v)
136    {
137       // создание тега, если queryEditText и tagEditText не пусты
```

```
138        if (queryEditText.getText().length() > 0 &&
139           tagEditText.getText().length() > 0)
140        {
141        makeTag(queryEditText.getText().toString(),
142           tagEditText.getText().toString());
143        queryEditText.setText(""); // очистка queryEditText
144        tagEditText.setText("");   // очистка tagEditText
145
146        // скрытие виртуальной клавиатуры
147        ((InputMethodManager) getSystemService(
148           Context.INPUT_METHOD_SERVICE)).hideSoftInputFromWindow(
149           tagEditText.getWindowToken(), 0);
150    }   // конец блока if
151    else // отображение сообщения о необходимости ввода
             // пользователем запроса и тега
152    {
153         // создание нового AlertDialog Builder
154         AlertDialog.Builder builder =
155            new AlertDialog.Builder(FavoriteTwitterSearches.this);
156
157         builder.setTitle(R.string.missingTitle); // название строки
158
159         // кнопка OK, скрывающая диалоговое окно
160         builder.setPositiveButton(R.string.OK, null);
161
162         // отображаемое сообщение
163         builder.setMessage(R.string.missingMessage);
164
165         // создание AlertDialog на основе AlertDialog.Builder
166         AlertDialog errorDialog = builder.create();
167         errorDialog.show(); // display the Dialog
168  } // конец блока else
169  } // конец метода onClick
170  };// конец анонимного внутреннего класса OnClickListener
171
```

Диалоговые окна часто включают несколько кнопок. В рассматриваемом случае требуется лишь одна кнопка, с помощью которой пользователь может ознакомиться с сообщением. В данном случае идет речь о «позитивной» кнопке (строка 160). Метод setPositiveButton принимает название кнопки (указано в строковом ресурсе R.string.OK) и ссылку на обработчик событий кнопки. В этом диалоговом окне не нужно отвечать на событие, поэтому для обработчика событий указан null. Как только пользователь касается кнопки, диалоговое окно исчезает с экрана.

В строке 163 настраивается метод, который отображается в диалоговом окне (указан с помощью строкового ресурса R.string.missingMessage). В строке 166 создается диалоговое окно AlertDialog путем вызова метода create для AlertDialog.Builder. В строке 167 вызывается метод show объекта AlertDialog, отображающий модальное диалоговое окно.

Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события clearTagsButton

В строках 173–213 из листинга 5.14 создается объект анонимного внутреннего класса clearTagsButtonListener, реализующий интерфейс OnClickListener. В строке 56 этот объект регистрируется как обработчик событий clearTagsButtons. В строках 175–212 реализуется метод onClick интерфейса OnClickListener. Этот метод отображает диалоговое окно AlertDialog, в котором пользователю предлагается подтвердить необходимость удаления всех сохраненных поисков.

Листинг 5.14. Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события clearTagsButton

```
172 // очистка всех сохраненных поисков
173 public OnClickListener clearTagsButtonListener = new OnClickListener()
174 {
175    @Override
176    public void onClick(View v)
177    {
178       // создание нового AlertDialog Builder
179       AlertDialog.Builder builder =
180          new AlertDialog.Builder(FavoriteTwitterSearches.this);
181
182       builder.setTitle(R.string.confirmTitle); // строка заголовка
183
184       // кнопка OK, которая скрывает диалоговое окно
185       builder.setPositiveButton(R.string.erase,
186          new DialogInterface.OnClickListener()
187          {
188             @Override
189             public void onClick(DialogInterface dialog, int button)
190             {
191                clearButtons(); //очистка сохраненных поисков из карты
192
193                // SharedPreferences.Editor для очистки поисков
194                SharedPreferences.Editor preferencesEditor =
195                   savedSearches.edit();
196
197                preferencesEditor.clear(); //удаление пар тег/запрос
198                preferencesEditor.apply(); // подтверждение изменений
199             }   // конец метода onClick
200          }   // конец анонимного внутреннего класса
201      ); // конец вызова метода setPositiveButton
202
203      builder.setCancelable(true);
204      builder.setNegativeButton(R.string.cancel, null);
205
206      // настройка отображаемого сообщения
207      builder.setMessage(R.string.confirmMessage);
208
209      // создание диалогового окна AlertDialog из AlertDialog.Builder
```

```
210        AlertDialog confirmDialog = builder.create();
211        confirmDialog.show(); // отображение диалогового окна
212     }  // конец метода onClick
213  }; // конец анонимного внутреннего класса OnClickListener
214
```

В строках 185–201 определяется «позитивная» кнопка AlertDialog и соответствующий этой кнопке обработчик событий. После щелчка на этой кнопке вызывается этот обработчик событий. В строке 191 вызывается объект clearButtons (см. листинг 5.12), который удаляет все объекты Button, представляющие сохраненные поисковые запросы. Затем вызывается объект SharedPreferences.Editor для savedSearches (строки 194–195), очищаются все пары «ключ/значение» путем вызова метода clear объекта SharedPreferences.Editor (строка 192). Затем подтверждаются изменения в файле (строка 198). Оператор в строке 203 «говорит» о том, что отображение диалогового окна может быть отменено пользователем. А именно, чтобы скрыть диалоговое окно, пользователю достаточно нажать кнопку Back (Назад). В строке 204 настраивается «отрицательная» кнопка диалогового окна и обработчик событий. Подобно «положительной» кнопке из листинга 5.13, эта кнопка просто скрывает диалоговое окно. В строках 207–211 настраивается сообщение диалогового окна, создается диалоговое окно, в котором отображается сообщение.

Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события, генерируемые newTagButton

В строках 216–234 листинга 5.15 создается объект анонимного внутреннего класса queryButtonListener, который реализует интерфейс OnClickListener. В строке 113 этот объект регистрируется в качестве объекта обработки событий для каждого из созданных объектов newTagButton.

В строках 218–233 реализуется метод onClick интерфейса OnClickListener. В строке 222 производится получение текста кнопки Button, на которой был произведен щелчок мышью, а в строке 223 выбирается соответствующий поисковый запрос из savedSearches. В строке 226 вызывается унаследованный метод getString объекта Activity. В результате обеспечивается получение строкового ресурса searchURL, который включает URL-ссылку страницы поиска Твиттера. Потом запрос присоединяется в конец URL-ссылки.

Листинг 5.15. Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события queryButton

```
215   // загрузка выбранного поиска в окно веб-браузера
216   public OnClickListener queryButtonListener = new OnClickListener()
217   {
218      @Override
219      public void onClick(View v)
220      {
221         // получение запроса
222         String buttonText = ((Button)v).getText().toString();
223         String query = savedSearches.getString(buttonText, null);
224
225         // создание URL-ссылки, соответствующей запросу нажатой кнопки
226         String urlString = getString(R.string.searchURL) + query;
```

продолжение ☞

Листинг 5.15 *(продолжение)*

```
227
228          // создание интента для запуска веб-браузера
229          Intent getURL = new Intent(Intent.ACTION_VIEW,
230             Uri.parse(urlString));
231
232          startActivity(getURL); // вызов интента
233       }   // конец метода onClick
234    }; // конец анонимного внутреннего класса OnClickListener
235
```

В строках 229–230 создается новый объект Intent (Интент), который будет использован для запуска веб-браузера устройства и отображения результатов поиска в Твиттере. Объект Intent представляет описание *действия*, которое будет выполняться по отношению к связанным *данным*. Первый аргумент, передаваемый конструктору Intent, представляет собой константу, описывающую выполняемое действие. В данный момент используется константа Intent.ACTION_VIEW, поскольку нужно отображать представление данных. Многие константы определены в классе Intent и описывают такие действия, как *поиск*, *выбор*, *отсылка* и *воспроизведение*. Второй аргумент (строка 230) — это Uri (uniform resource identifier, унифицированный идентификатор ресурса). Этот аргумент указывает на данные, по отношению к которым выполняется действие. Метод parse из класса Uri преобразует строку, представляющую URL-ссылку (uniform resource locator, унифицированный локатор ресурса), в Uri-ссылку.

В строке 232 передается Intent методу startActivity (косвенно наследуется из класса Context), который запускает корректный класс Activity, реализующий выполнение определенного действия по отношению к текущим данным. В рассматриваемом случае указан просмотр URI-ссылки, поэтому Intent запускает веб-браузер устройства для отображения соответствующей веб-страницы. На этой странице отображаются результаты поиска, выполненного в Твиттере.

А теперь рассмотрим пример **неявного объекта Intent**. При определении подобного объекта не указывается явно компонент, отображающий веб-страницу, а просто разрешается системе запускать наиболее подходящую деятельность, основываясь на типе данных. Если для выполнения этой задачи могут использоваться несколько деятельностей, а данные передаются startActivity, система отображает диалоговое окно, в котором пользователь может выбрать используемую деятельность. Если система не может найти подходящую деятельность, метод startActivity генерирует исключение ActivityNotFoundException. Поэтому рекомендуется принять меры по обработке этого исключения. В данном случае этого не делается, поскольку на устройствах Android, на которых выполняется это приложение, установлен браузер, способный отображать веб-страницы.

При разработке приложений в будущем будут использованы **явные Intents**, в которых точно указывается класс Activity, выполняемый в том же приложении. Список приложений и поддерживаемых интентов можно найти на веб-сайтах:

- openintents.org;
- developer.android.com/guide/appendix/g-app-intents.html.

Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события editButton

В строках 237–253 листинга 5.16 создается объект анонимного внутреннего класса editButtonListener, который реализует интерфейс OnClickListener. В строке 118 этот объект регистрируется в качестве каждого из объектов обработки событий newEditButton. В строках 239–252 реализуется метод onClick интерфейса OnClickListener. Чтобы определить, каким образом будет редактироваться запрос кнопки поиска, сначала получим *родительский макет* editButton (строка 243). Именно здесь находится компонент editButton, который используется для получения объекта Button с ID, равным R.id.newTagButton в этом макете (строки 244–245). По сути это и есть соответствующая кнопка поиска. В строке 247 получаем текст searchButton, который используется в строке 250 для установки значения tagEditText. И наконец, в строке 251 получаем соответствующий запрос из объекта savedSearches и отображаем его в queryEditText.

Листинг 5.16. Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события editButton

```
236   // изменение выбранного поискового запроса
237   public OnClickListener editButtonListener = new OnClickListener()
238   {
239      @Override
240      public void onClick(View v)
241      {
242         // получение всех необходимых компонентов GUI
243         TableRow buttonTableRow = (TableRow) v.getParent();
244         Button searchButton =
245            (Button) buttonTableRow.findViewById(R.id.newTagButton);
246
247         String tag = searchButton.getText().toString();
248
249         // EditTexts должны соответствовать выбранному тегу и запросу
250         tagEditText.setText(tag);
251         queryEditText.setText(savedSearches.getString(tag, null));
252      }  // конец метода onClick
253   }; // конец анонимного внутреннего класса OnClickListener
254 } // конец класса FavoriteTwitterSearches
```

5.6. Файл AndroidManifest.xml

В процесс создания любого проекта Android в среде Eclipse модуль ADT Plugin создает и конфигурирует файл AndroidManifest.xml (также известный под названием *манифест приложения*). В этом файле находится информация о приложении. В этом разделе будет рассмотрено содержимое этого файла (листинг 5.17), а также новое, добавленное в этот файл свойство. Другие свойства файла манифеста будут рассматриваться по мере возникновения необходимости в них. Дополнительные сведения о файле манифеста можно найти на следующем веб-сайте: developer.android.com/guide/topics/manifest/manifest-intro.html.

Листинг 5.17. Файл AndroidManifest.xml для приложения Favorite Twitter Searches

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <manifest xmlns:android=http://schemas.android.com/apk/res/android
 3    package="com.deitel.favoritetwittersearches"
 4    android:versionCode="1" android:versionName="1.0">
 5    <application android:icon="@drawable/icon"
 6       android:label="@string/app_name">
 7       <activity android:name=".FavoriteTwitterSearches"
 8          android:label="@string/app_name"
 9          android:windowSoftInputMode="stateAlwaysHidden">
10          <intent-filter>
11             <action android:name="android.intent.action.MAIN" />
12             <category android:name="android.intent.category.LAUNCHER" />
13          </intent-filter>
14       </activity>
15    </application>
16    <uses-sdk android:targetSdkVersion="10" android:minSdkVersion="8"/>
17  </manifest>
```

Элемент manifest (строки 2–17) — это корневой элемент в AndroidManifest.xml. Атрибут элемента package (строка 3) определяет пакет, который используется для управления кодом. Атрибут элемента android:versionCode (строка 4) определяет внутренний целочисленный номер версии, который используется приложением для сравнения версий различных приложений. Атрибут элемента android:versionName (строка 4) определяет номер версии, который видят пользователи при управлении приложением, установленным на устройстве.

Внутри элемента manifest находятся элементы nested application (строки 5–15) и uses-sdk (строка 16). Элемент application обязателен для указания. Атрибут элемента android:icon определяет ресурс drawable, используемый в качестве пиктограммы приложения. Если пользователь не определит собственную пиктограмму, приложение будет использовать стандартную пиктограмму ADT Plugin при создании проекта приложения. Версии этой пиктограммы хранятся в папках приложения res/drawable. С помощью атрибута элемента android:label определяется название приложения. Элемент uses-sdk определяет библиотеку SDK, используемую приложением (10 соответствует Android SDK версии 2.3.3), и минимальную версию SDK (8 соответствует версии 2.2). Эти настройки обеспечивают выполнение приложения на устройствах Android 2.2 и выше.

Внутри элемента application находится элемент activity (строки 7–14), который содержит информацию о классе Activity приложения. Если приложение имеет более одного класса Activity, каждому из них будет соответствовать свой собственный элемент activity. Атрибут android:name (строка 7) определяет полностью квалифицированное имя класса Activity. Если имя класса предварить точкой (.), оно автоматически добавится к имени пакета, указанному с помощью элемента manifest. Атрибут android:label (строка 8) определяет строку, которая отображается вместе с Activity. По умолчанию манифест сконфигурирован таким образом, что в качестве значения этого атрибута используется имя приложения. В строку 9 мы добавили атрибут android:windowSoftInputMode. Значение stateAlwaysHidden определяет, будет ли отображаться виртуальная клавиатура после запуска Activity. Чтобы добавить этот атрибут, нужно либо изменить XML-код непосредственно, либо дважды щелкнуть мышью на файле AndroidManifest.xml проекта,

чтобы открыть редактор манифеста. На рис. 5.8 показана вкладка Application (Приложение), выбранная в окне редактора манифеста. Названия вкладок отображаются в нижней части окна редактора. Чтобы установить атрибут android:windowSoftInputMode, выберите параметр .FavoriteTwitterSearches в разделе Application Nodes (Узлы приложения) окна (в нижней левой части). В правой нижней части окна редактора отобразятся атрибуты элементов activity. Выберите атрибут Window soft input mode (Режим программного ввода в окне) и щелкните на кнопке Select... (Выбрать...). На экране появится список доступных параметров. Выберите атрибут stateAlwaysHidden и щелкните на кнопке OK.

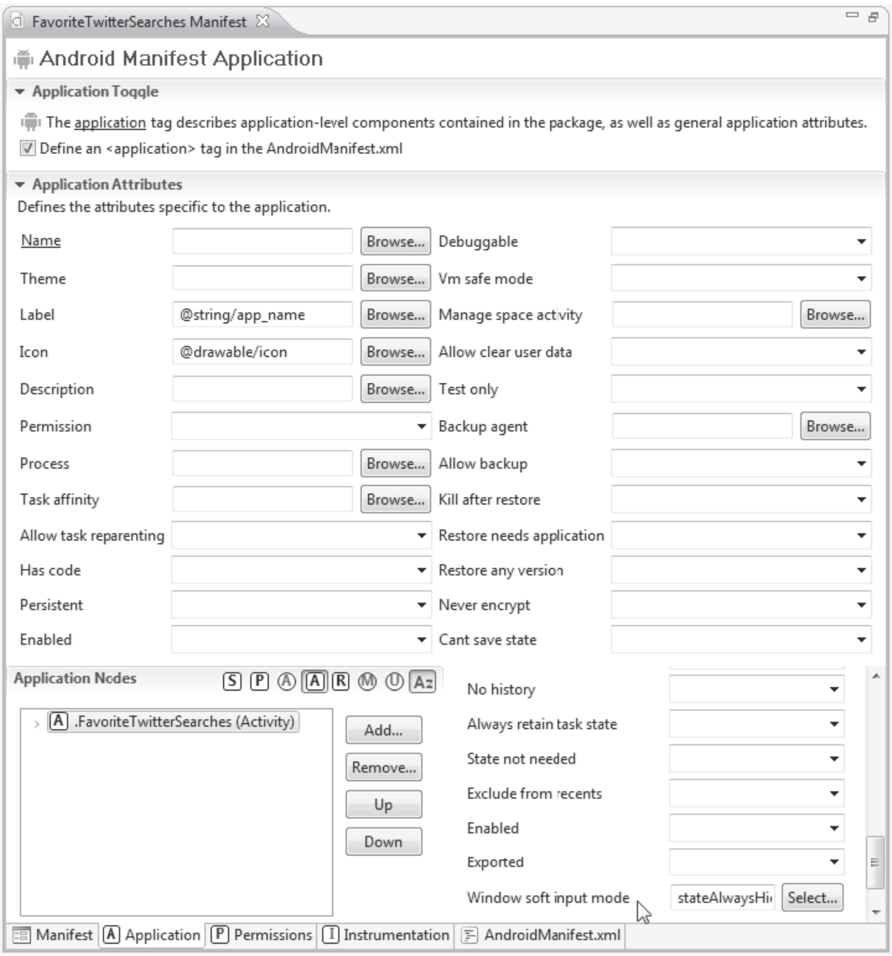


Рис. 5.8. Вкладка Application в окне редактора манифеста

Внутри элемента activity находится элемент intent-filter (строки 10–13), который определяет типы интентов, на которые может реагировать класс Activity. Этот элемент должен включать один или большее число элементов action. Элемент в строке 11 — основная деятельность приложения, которая отображается после запуска приложения.

Элемент category (строка 12) определяет тип компонента Android, обрабатывающего событие. В данном случае значение "android.intent.category.LAUNCHER" свидетельствует о том, что эта деятельность должна быть внесена в список модулем запуска приложения вместе с другими приложениями устройства.

5.7. Резюме

В этой главе мы рассматривали приложение Favorite Twitter Searches. На первом этапе был создан графический интерфейс пользователя (GUI). Вы познакомились с компонентом ScrollView, который представляет собой группу ViewGroup, позволяющую выполнять прокрутку содержимого, не помещающегося на экране. Этот компонент также позволяет отображать сколь угодно длинные списки сохраненных поисковых запросов. Каждый запрос имеет связанную кнопку (Button), после касания которой выполняется поиск в окне веб-браузера устройства. Вы также узнали о том, каким образом можно создавать файлы ресурсов в диалоговом окне New Android XML File, а именно — файл colors.xml — хранилище ресурсов цвета, файл dimen.xml, предназначенный для хранения размеров, и файл второго макета, на основе которого методом «раздувания» был получен макет. Были рассмотрены методы ссылки на цвета и размеры в XML-разметке и порядок использования заранее определенных цветов из класса Android R.color.

Поисковые пары «тег/запрос» были сохранены в файле SharedPreferences, связанном с приложением. Мы рассмотрели скрытие виртуальной клавиатуры, выполненное программным путем. С помощью объекта SharedPreferences.Editor было выполнено сохранение значений, изменение значений, а также удаление значений из файла SharedPreferences. В ответ на касание пользователем кнопки поиска загружались Uri-ссылки в окно веб-браузера устройства путем создания нового интента с дальнейшей передачей методу startActivity из класса Context.

Объекты AlertDialog.Builder были применены для конфигурирования и создания окон AlertDialogs, в которых отображаются сообщения для пользователя. Было продемонстрировано программное создание компонентов GUI путем ручного «раздувания» файла XML-разметки. В результате приложение получило возможность динамического изменения GUI на основе взаимодействия с пользователем. Эта методика была использована для создания компонента TableRow, включающего два новых объекта Button (кнопки), один из которых выполняет поиск, а второй позволяет изменить поиск. Эти компоненты TableRows были добавлены в TableLayout в ScrollView, в результате чего все тегированные поиски могут отображаться в области прокрутки экрана.

И наконец, был рассмотрен файл AndroidManifest.xml и показано, как сконфигурировать приложение, чтобы после запуска приложения не отображалась виртуальная клавиатура.

В главе 6 будет создано приложение Flag Quiz Game, на экране которого отображается флаг определенной страны, название которой предстоит выбрать пользователю среди доступных 3, 6 или 9 вариантов. С помощью меню и флажков вы сможете настроить приложение, ограничив количество отображаемых флагов и стран определенными регионами мира.

6

Приложение Flag Quiz Game

Ресурсы, AssetManager, анимация с переходами, обработчик, меню и регистрация сообщений об ошибках

В этой главе...

- Хранение строковых массивов в файле strings.xml.
- Хранение набора изображений в подпапках с помощью папки ресурсов.
- Отображение списка ресурсов приложения с помощью AssetManager.
- Выбор различных флагов с помощью генератора случайных чисел.
- Отображение флага в компоненте ImageView с помощью объекта Drawable.
- Планирование выполняемых в будущем действий с помощью Handler.
- Хранение коллекций элементов и пар «имя-значение» с помощью объектов ArrayList и HashMap соответственно.
- Создание объектов Menu и MenuItems, позволяющих конфигурировать параметры приложения, с помощью переопределенного метода onCreateOptionsMenu класса Activity.
- Использование механизма регистрации Android для регистрации сообщений об ошибках.

6.1. Введение

Приложение Flag Quiz Game применяется для проверки знания пользователем флагов различных стран (рис. 6.1). После запуска приложения на экране появляется изображение флага и три варианта возможного ответа. Один из вариантов является правильным,

а остальные — неправильные — не повторяющиеся и случайным образом выбранные. Приложение иллюстрирует процесс игры, отображая номер вопроса (из 10 возможных) в `TextView`, находящемся над изображением текущего флага.

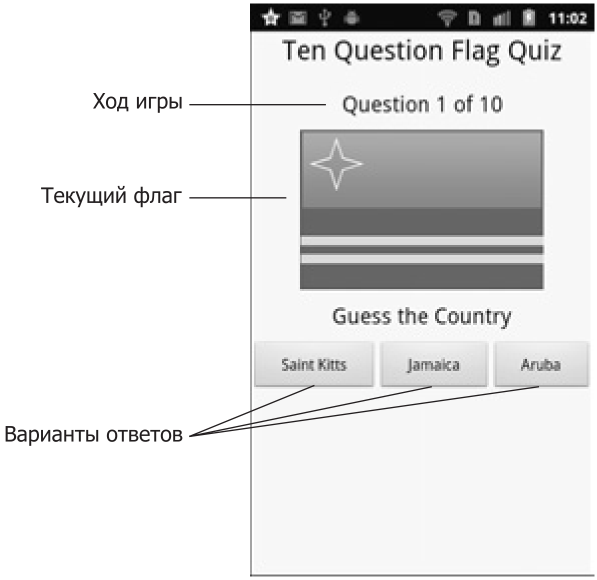


Рис. 6.1. Приложение Flag Quiz Game



Рис. 6.2. Корректный вариант отображается на экране в случае его выбора пользователем

Выбран правильный вариант ответа

Пользователь выбирает страну, касаясь соответствующего компонента Button. Если выбран правильный вариант ответа, приложение скрывает все компоненты Buttons, а в нижней части экрана отображается название страны, выделенное зеленым цветом, за которым следует восклицательный знак (рис. 6.2). После секундной задержки приложение загружает следующий флаг и отображает новый набор компонентов Buttons, определяющих варианты ответов.

Выбран неправильный ответ

Если пользователь выбрал неправильный вариант ответа, приложение деактивирует компонент Button, соответствующий выбранной стране, с помощью анимации «встряхивает» флаг и отображает в нижней части экрана красным цветом сообщение Incorrect! (рис. 6.3). После этого пользователь продолжает выбирать страну до тех пор, пока не угадает.

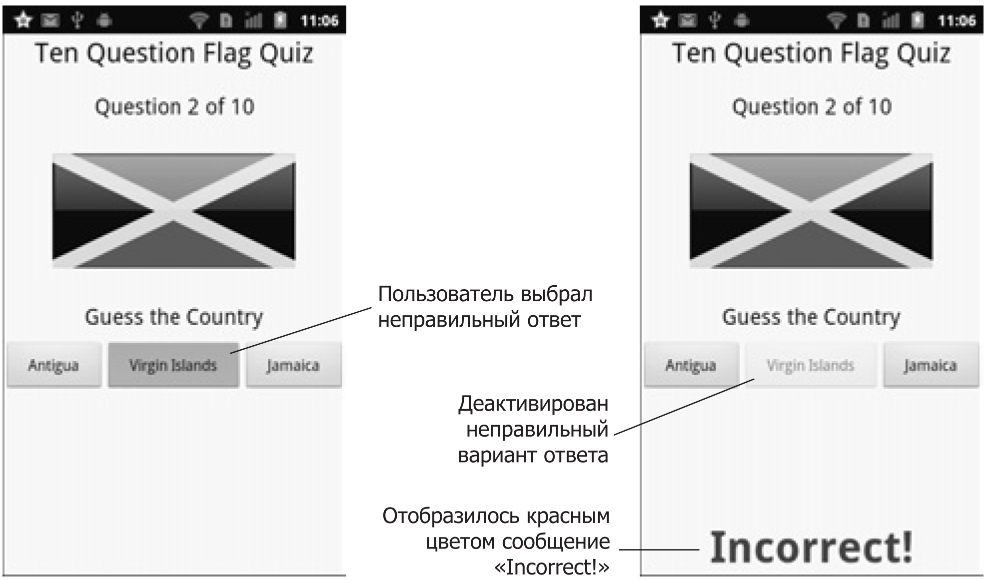


Рис. 6.3. Деактивированый неправильный ответ в окне приложения Flag Quiz Game

После 10 правильных ответов

После 10 корректных ответов появляется всплывающее окно AlertDialog, в котором отображается общее количество предпринятых пользователем попыток и процент правильных ответов (рис. 6.4). Если пользователь коснется кнопки Reset Quiz, начнется новый сеанс игры, в котором используются текущие настройки.

Настройка количества вариантов ответов для каждого флага

С помощью меню приложения пользователь может произвести настройку викторины. После касания кнопки меню устройства отображаются параметры меню Select Number

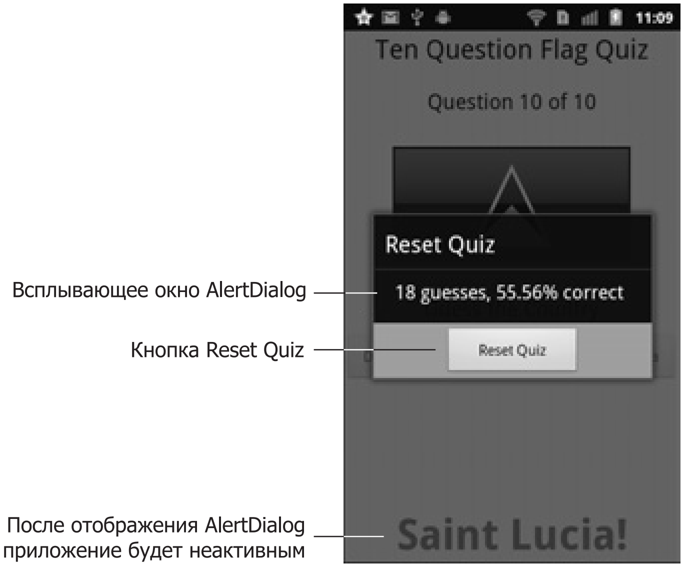


Рис. 6.4. Результаты игры

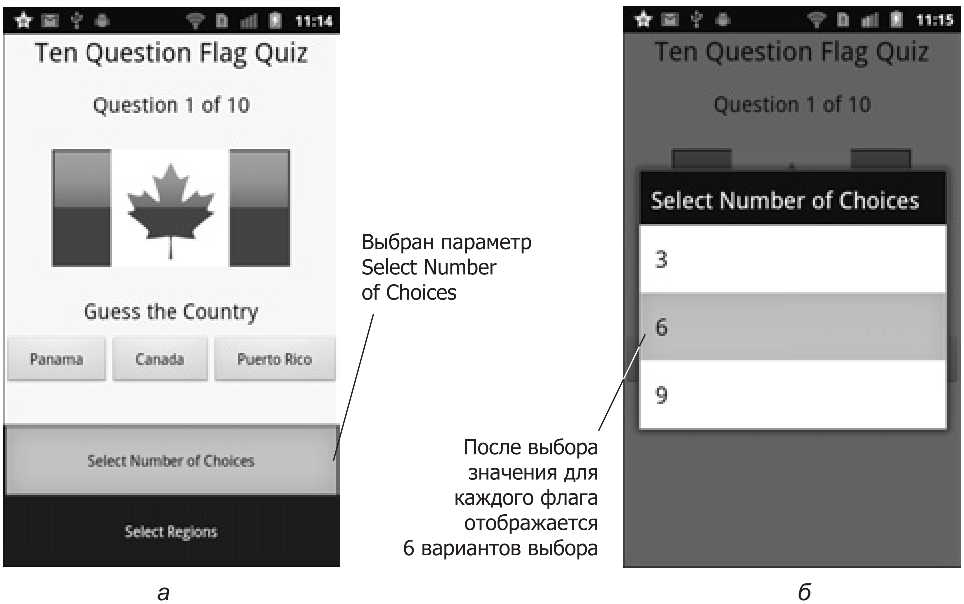
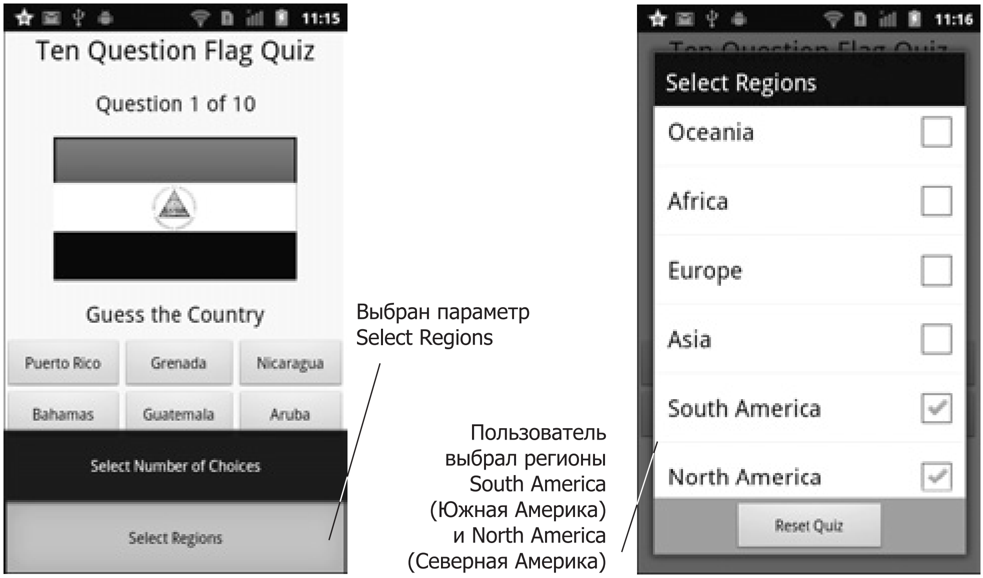


Рис. 6.5. Меню приложения Flag Quiz Game: *а* — меню, отображаемое после выбора параметра Select Number of Choices; *б* — во всплывающем окне AlertDialog отображается количество вариантов выбора ответов

of Choices (Выбрать количество вариантов) и Select Regions (Выбрать регион). После выбора параметра меню Select Number of Choices появится всплывающее окно AlertDialog, в котором можно выбрать 3, 6 или 9 вариантов ответа, отображаемых под каждым фла-



а б

Рис. 6.6. Диалоговое окно выбора регионов приложения Flag Quiz Game: *а* — меню, в котором выбирается параметр Select Regions; *б* — во всплывающем окне AlertDialog показаны выбранные регионы

гом (рис. 6.5). После выбора одного из этих параметров будет выполнен перезапуск игры с учетом выбранного количества вариантов ответа для каждого флага (и ранее выбранными регионами мира).

Настройка регионов, для которых выбираются флаги

Если в меню приложения выбрать параметр Select Regions, приложение отображает всплывающее окно AlertDialog, в котором находятся флажки для каждого региона мира, — пять флажков для континентов и шестой флажок для Океании, включающей Австралию, Новую Зеландию и тихоокеанские острова (рис. 6.6). После выбора соответствующих регионов в викторине будут использованы флаги относящихся к этим регионам стран. После выбора кнопки Reset Quiz произойдет перезапуск игры, причем будут выбраны флаги стран, относящихся к выбранным регионам.

6.2. Тестирование приложения Flag Quiz Game

Открытие и выполнение приложения

Откройте среду Eclipse и импортируйте проект приложения Flag Quiz Game. Выполните следующие действия:

1. *Отройте диалоговое окно* Import. Чтобы открыть диалоговое окно Import, выполните команды File ▸ Import... (Файл ▸ Импорт...).

2. *Импортируйте проект приложения* FlagQuiz Game. В диалоговом окне Import раскройте узел General (Общий) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область). Затем щелкните на кнопке Next > (Далее >) для выполнения шага Import Projects (Импорт проектов). Выберите корневой каталог и щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папки) выберите папку FlagQuizGame в папке примеров книги и щелкните на кнопке OK. Щелкните на кнопке Finish (Готово) для импорта проекта в среду Eclipse. Проект отобразится в окне Package Explorer, находящемся в левой части окна Eclipse.
3. *Запустите приложение FlagQuiz Game*. В среде Eclipse щелкните правой кнопкой мыши на проекте FlagQuizGame, находящемся в окне Package Explorer, затем в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android).

Настройка викторины

Коснитесь кнопки Menu (либо кнопки меню устройства) для получения доступа к меню, в котором можно просмотреть параметры приложения. Чтобы выбрать количество вариантов ответа, отображаемых для каждого флага (см. рис. 6.5), коснитесь кнопки Select Number of Choices. По умолчанию после первого запуска приложения отображаются три варианта ответа для каждого флага. Чтобы отобразить шесть вариантов ответов для каждого флага, коснитесь параметра 6.

Коснитесь кнопки Select Regions, чтобы отобразить набор флажков, представляющих регионы мира (см. рис. 6.6). По умолчанию после первого запуска приложения отображаются все регионы, поэтому для викторины случайным образом выбираются флаги, относящиеся ко всем регионам мира. Коснитесь флажков Africa (Африка) и Oceania (Океания), чтобы отключить их. В результате флаги, относящиеся к соответствующим регионам, будут исключены из викторины. Чтобы начать новую игру с изменившимися настройками, коснитесь кнопки Reset Quiz (Перегрузить викторину).

Завершение викторины

После начала новой викторины доступны шесть вариантов ответа для каждого флага, причем при выборе флагов исключены регионы Africa и Oceania. В процессе игры после отображения флага коснитесь страны, которой (по вашему мнению) принадлежит этот флаг. Если вы не угадали, предпринимайте повторные попытки до тех пор, пока не достигнете успеха. После успешного отгадывания стран, которым принадлежат 10 отображаемых флагов, окно викторины станет неактивным, и отобразится всплывающее окно AlertDialog, в котором будет указано количество попыток отгадывания и процент правильных ответов (см. рис. 6.4). Чтобы начать новую игру, нажмите кнопку Reset Quiz.

6.3. Обзор применяемых технологий

Использование папки приложения assets

Приложение включает по одному изображению для каждого флага[1]. Эти изображения загружаются приложением в случае необходимости в них. Они находятся в папке assets

[1] Изображения флагов были получены с веб-сайта www.free-country-flags.com.

приложения (в эту папку были перетащены папки всех регионов, находящиеся в файловой системе). Данные папки находятся в папке images/FlagQuizGameImages, находящейся в папке примеров книги. В отличие от папки приложения drawable, в которой графический контент должен находиться на корневом уровне в каждой папке, папка assets может включать файлы любого типа, которые могут быть организованы в виде подпапок (изображения флагов стран, относящихся к одному региону, находятся в отдельной подпапке). Доступ к файлам, находящимся в папках assets, обеспечивается с помощью диспетчера AssetManager (пакет android.content.res), который может поддерживать список всех имен файлов в отдельной подпапке папки assets, а также может применяться для доступа к каждому отдельному ресурсу.

Для отображения флага, относящегося к вопросу викторины, с помощью AssetManager открывается поток InputStream (пакет java.io) для считывания содержимого графического файла флага. Затем этот поток используется в качестве аргумента статического метода createFromStream класса Drawable, с помощью которого создается объект Drawable. Этот объект Drawable (пакет android.graphics.drawable) затем используется в качестве компонента ImageView, формирующего отображение с помощью метода setImageDrawable класса ImageView.

Обеспечение доступа к параметрам приложения с помощью меню

Количество отображаемых вариантов ответов и регионов, в которых могут быть выбраны флаги, настраивается пользователем с помощью объекта приложения Menu (пакет android.view). Для задания параметров меню выполняется переопределение метода onCreateOptionsMenu класса Activity, а также добавляются параметры в Menu, которые вышеупомянутый метод принимает в качестве аргумента. При выборе элемента в меню вызывается метод onOptionsItemSelected класса Activity. Для отображения соответствующих параметров в AlertDialogs выполняется переопределение этого метода.

Использование объекта Handler для вызова Runnable с задержкой

Для определения задержки перед отображением следующего флага после выбора корректного варианта ответа используется объект Handler (пакет android.os). С помощью этого объекта реализована миллисекундная задержка перед вызовом Runnable. Метод postDelayed объекта Handler принимает в качестве аргументов Runnable (для вызова) и задержку, выраженную в миллисекундах.

Анимация флага в случае неправильного ответа

Если пользователь выберет неправильный ответ, приложение «встряхивает» флаг путем применения методов класса Animation (пакет android.view.animation) по отношению к компоненту ImageView. С помощью статического метода loadAnimation класса AnimationUtils выполняется загрузка анимации из XML-файла, где хранятся параметры анимации. Количество повторов анимации определяется с помощью метода setRepeatCount класса Animation. Выполнение же самой анимации в ImageView реализуется путем вызова метода startAnimation класса View (причем в качестве аргумента используется Animation) для компонента ImageView.

Регистрация исключений с помощью Log.e

Регистрация исключений (в целях выполнения отладки) осуществляется с помощью встроенного в Android механизма регистрации. При этом для краткосрочного хранения сообщений об исключениях используется циркулярный буфер. В операционной системе Android поддерживается класс Log (пакет android.util), включающий множество статических методов, которые представляют различные аспекты сообщений об исключениях. Для просмотра зарегистрированных сообщений применяется инструмент Android logcat. Эти сообщения также отображаются на вкладке LogCat перспективы Android DDMS (Dalvik Debug Monitor Server) в Eclipse. Дополнительные сведения о регистрации сообщений можно найти на веб-сайте developer.android.com/reference/android/util/Log.html/.

Структуры данных Java

При создании приложения из этой главы использованы различные структуры данных из пакета java.util. Приложение в динамическом режиме загружает названия графических файлов и хранит их в массиве ArrayList<String>. С помощью метода shuffle класса Collections случайным образом изменяется порядок следования имен файлов в массиве ArrayList<String> для каждой новой игры. Второй массив ArrayList<String> используется для хранения имен файлов изображений, соответствующих 10 странам для текущей викторины. Объект HashMap<String, Boolean> используется для хранения названий регионов и соответствующих булевых значений, определяющих использование соответствующего региона. Ссылки на объекты ArrayList<String> и HashMap<String, Boolean> реализованы с помощью переменных, имеющих типы интерфейса List<String> и Map<String, Boolean> соответственно. Это хорошая практика программирования на Java, которая позволяет легко изменять структуры данных, не затрагивая при этом остальной код приложения. Кроме того, для ссылки на ключи объекта HashMap используется интерфейс Set<String>.

6.4. Создание графического интерфейса приложения и файлов ресурсов

В этом разделе рассматривается создание графического интерфейса пользователя для приложения Flag Quiz Game. Будет сформирована дополнительная XML-разметка, которая путем динамического «раздувания» создает компоненты Buttons с именами стран, представляющими каждый возможный ответ на викторину. Мы также создадим XML-представление *«трясущейся» анимации*, применяемой к изображению флага в случае неправильного ответа пользователя.

6.4.1. Компонент main.xml LinearLayout

При разработке этого приложения используется «вертикальный» макет LinearLayout из main.xml. На рис. 6.7 показаны названия компонентов графического интерфейса пользователя. Согласно используемому в книге соглашению о наименовании в свойстве Id каждого компонента в XML-разметке и каждом имени переменной в коде Java используется имя класса компонента GUI.

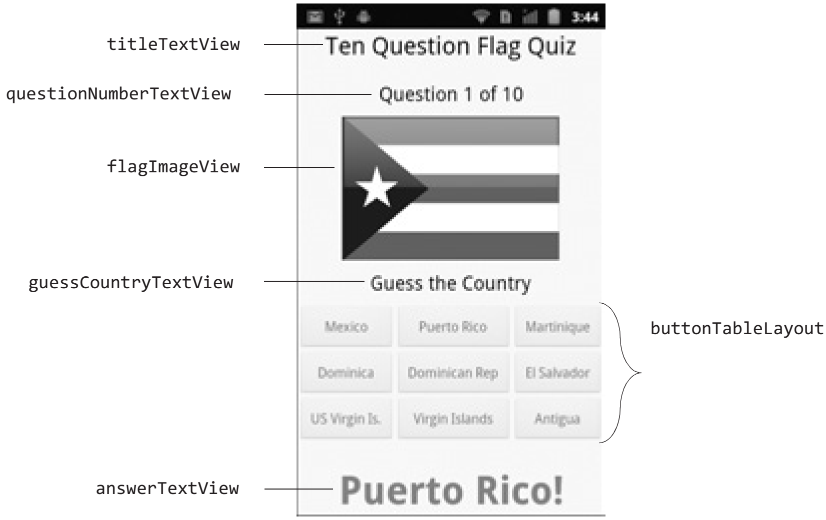


Рис. 6.7. Компоненты GUI приложения Flag Quiz Game, подписанные с помощью значений свойства Id. Компонент buttonTableLayout включает компоненты tableRow0, tableRow1 и tableRow2, компоненты Buttons генерируются динамически на основе количества отображаемых вариантов ответов

6.4.2. Создание проекта

Начните создание нового проекта Android под названием FlagQuizGame. В диалоговом окне New Android Project (Новый проект Android) введите перечисленные далее значения, а затем нажмите кнопку Finish (Готово).

- Build Target (Операционная система): Android 2.3.3.
- Application name (Имя приложения): FlagQuizGame.
- Package name (Название пакета): com.deitel.flagquizgame.
- Create Activity (Создать деятельность): FlagQuizGame.
- Min SDK Version (Минимальная версия SDK): 8.

6.4.3. Создание и редактирование файлов ресурсов

Как и в случае с приложением из предыдущей главы, создайте файлы colors.xml и dimen.xml, в которых хранятся литеральные значения цвета и размера соответственно. Чтобы создать каждый из этих файлов, выполните следующие действия:

1. Щелкните правой кнопкой мыши на имени проекта в окне Package Explorer и выполните команды New ▸ Other... (Создать ▸ Другой...), затем в диалоговом окне New (Создать) в узле Android выберите параметр Android XML File (XML-файл Android). На экране появится диалоговое окно New Android XML File (Создать XML-файл Android).
2. В текстовое поле File (Файл) введите имя colors.xml.

3. В разделе What type of resource would you like to create? (Укажите тип создаваемого ресурса) выберите переключатель Values (Значения). После этого в папке проекта res/values появится новый файл.
4. Для завершения создания файла щелкните на кнопке Finish (Готово).
5. Повторите описанный выше процесс для создания файла dimen.xml.

Содержимое файлов colors.xml и dimen.xml приведено в листингах 6.1–6.2. Определенные с помощью этих файлов цвета и размеры используются в файле разметки main.xml. Добавьте эти ресурсы в файлы вашего проекта.

Листинг 6.1. Цвета, определенные в файле colors.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
   <color name="text_color">#000000</color>
   <color name="background_color">#FFFFCC</color>
   <color name="correct_answer">#00CC00</color>
   <color name="incorrect_answer">#FF0000</color>
</resources>
```

Листинг 6.2. Размеры, определенные в файле dimen.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
   <dimen name="title_size">25sp</dimen>
   <dimen name="flag_width">227dp</dimen>
   <dimen name="flag_height">150dp</dimen>
   <dimen name="answer_size">40sp</dimen>
   <dimen name="text_size">20sp</dimen>
</resources>
```

strings.xml

Как и при создании приложения в предыдущей главе, строковые ресурсы определяются в файле strings.xml (листинг 6.3). Также на первом этапе определяются два строковых массива в файле strings.xml. Эти массивы представляют названия регионов (строки 18–25) и кнопки Buttons, которые определяют количество вариантов ответов и отображаются для каждого вопроса (строки 26–30) соответственно. Эти конструкции можно задать непосредственно в XML с помощью элементов string-array и item (листинг 6.3).

Листинг 6.3. Строки, определенные в файле strings.xml

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <resources>
 3    <string name="app_name">FlagQuizGame</string>
 4    <string name="choices">Select Number of Choices</string>
 5    <string name="correct">correct</string>
 6    <string name="guess_country">Guess the Country</string>
 7    <string name="guesses">guesses</string>
 8    <string name="incorrect_answer">Incorrect!</string>
 9    <string name="more_regions_title">More Regions Required</string>
10    <string name="more_regions_message">There are not enough countries in
```

```
      the selected regions. Please select more regions.</string>
   <string name="of">of</string>
   <string name="ok">OK</string>
   <string name="question">Question</string>
   <string name="quiz_title">Ten Question Flag Quiz</string>
   <string name="regions">Select Regions</string>
   <string name="reset_quiz">Reset Quiz</string>
   <string-array name="regionsList">
      <item>Africa</item>
      <item>Asia</item>
      <item>Europe</item>
      <item>North_America</item>
      <item>Oceania</item>
      <item>South_America</item>
   </string-array>
   <string-array name="guessesList">
      <item>3</item>
      <item>6</item>
      <item>9</item>
   </string-array>
</resources>
```

Для создания перечисленных выше массивов можно воспользоваться редактором файлов ресурсов. Выполните следующие действия:

1. В окне редактора щелкните на кнопке Add... (Добавить...), в появившемся диалоговом окне выберите параметр String Array (Строковый массив) и щелкните на кнопке OK.
2. В поле Name (Имя), которое отображается в правой части окна редактора, укажите имя массива.
3. В списке ресурсов щелкните правой кнопкой мыши на имени массива, во всплывающем меню выберите параметр Add... (Добавить...), затем щелкните на кнопке OK, чтобы добавить новый элемент Item в массив.
4. Повторите шаг 3 для требуемого числа элементов массива.
5. Выберите каждый элемент Item в списке ресурсов и задайте его значение в поле Value (Значение) в правой части окна редактора.

6.4.4. Добавление компонентов в макет LinearLayout

Используя методики, изученные в предыдущих главах, создадим графический интерфейс пользователя, представленный на рис. 6.7. Начнем с формирования базового макета и элементов управления, затем завершим проект настройкой свойств элементов управления. В случае необходимости будут использованы ресурсы из файлов strings.xml (см. листинг 6.3), colors.xml (см. листинг 6.1) и dimen.xml (см. листинг 6.2). В этой главе вашему вниманию будет представлен обзор создания GUI приложения. При создании последующих приложений мы уделим внимание только новым особенностям и свойствам GUI, хотя завершенная XML-разметка будет представлена полностью, поэтому вы сможете просмотреть значения атрибутов для каждого компонента.

Шаг 1. Конфигурирование макета LinearLayout

В окне Outline (Структура) выберите макет LinearLayout и установите значения следующих свойств.

- Background (Фон): @color/background_color.
- Gravity (Выравнивание): center_horizontal.
- Id (Идентификатор): @+id/linearLayout.

Также измените значения свойств Layout width и Layout height, выбрав вместо fill_parent (не рекомендуется) match_parent.

Шаг 2. Добавление компонентов и конфигурирование их свойств

Используя рис. 6.7 в качестве руководства, добавьте компоненты TextViews, ImageView и TableLayout в макет linearLayout приложения. Установите значения свойств Id и Text для этих компонентов. Просмотрите XML-элементы в финальном файле main.xml (листинг 6.4), чтобы ознакомиться со значениями атрибутов каждого компонента. В листинге выделены важные особенности и ресурсы. Не создавайте объекты Buttons в TableRows, поскольку они генерируются динамически в процессе разгадывания викторины.

Листинг 6.4. XML-разметка (main.xml) приложения FlagQuizGame

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
   android:id="@+id/linearLayout" android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:gravity="center_horizontal"
   android:background="@color/background_color">

    <TextView android:id="@+id/titleTextView"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:text="@string/quiz_title"
          android:layout_marginBottom="10dp"
       android:textSize="@dimen/title_size"
       android:textColor="@color/text_color" android:gravity="center">
    </TextView>

    <TextView android:id="@+id/questionNumberTextView"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginBottom="10dp"
          android:layout_marginTop="10dp"
       android:textColor="@color/text_color"
       android:textSize="@dimen/text_size"
          android:layout_gravity="center"
       android:gravity="center"></TextView>

    <ImageView android:id="@+id/flagImageView"
```

```
27        android:adjustViewBounds="false"
28        android:layout_width="@dimen/flag_width"
29        android:layout_height= "@dimen/flag_height"></ImageView>
30
31     <TextView android:id="@+id/guessCountryTextView"
32        android:layout_width="wrap_content"
33        android:layout_height="wrap_content"
34        android:layout_marginBottom="10dp"
             android:layout_marginTop="10dp"
35        android:text="@string/guess_country"
36        android:textColor="@color/text_color"
37        android:textSize= "@dimen/text_size"></TextView>
38
39     <TableLayout android:id="@+id/buttonTableLayout"
40        android:layout_width="match_parent"
41        android:layout_height="wrap_content"
42        android:layout_weight="1" android:stretchColumns="0,1,2">
43        <TableRow android:id="@+id/tableRow0"
44           android:layout_width="match_parent"
45           android:layout_height="wrap_content"
46           android:orientation="horizontal"></TableRow>
47        <TableRow android:id="@+id/tableRow1"
48           android:layout_width="match_parent"
49           android:layout_height="wrap_content"
50           android:orientation="horizontal"></TableRow>
51        <TableRow android:id="@+id/tableRow2"
52           android:layout_width="match_parent"
53           android:layout_height="wrap_content"
54           android:orientation="horizontal"></TableRow>
55        </TableLayout>
56
57        <TextView android:id="@+id/answerTextView"
58           android:layout_width="match_parent"
59           android:layout_height="wrap_content"
60           android:textSize="@dimen/answer_size"
61           android:layout_gravity="center" android:textStyle="bold"
62           android:gravity="center"></TextView>
63  </LinearLayout>
```

Примечания к файлу main.xml

В строке 27 задается атрибут android:adjustViewBounds компонента ImageView, определяющий поддержку компонентом ImageView сохранения пропорций для класса Drawable. В рассматриваемом случае этому атрибуту присвоено значение false, позволяющее изменять размеры изображений флагов.

В строке 42 атрибуту android:layout_weight компонента buttonTableLayout присвоено значение 1. В результате компонент buttonTableLayout приобретает больший «вес» (по сравнению со всеми другими компонентами) при изменении размеров макета linearLayout на основе свободного пространства. Поскольку значение атрибута android:layout_weight указывается только для единственного компонента buttonTableLayout, этот компонент

«растягивается» по вертикали, заполняя пространство, которое не занято другими компонентами. Атрибуту android:stretchColumns компонента buttonTableLayout также присваиваются значения 0,1,2, определяющие «растяжение» всех трех столбцов данного компонента TableRow, занимающих доступное пространство в горизонтальном направлении.

6.4.5. Создание динамически «раздуваемой» кнопки

А теперь определим XML-разметку компонента Button. Приложение «раздувает» этот XML-файл, создавая компонент Button для каждого ответа. В разделе 6.5 будут сконфигурированы компоненты Buttons и связаны с соответствующим компонентом TableRow. Чтобы создать другой файл XML-разметки, выполните следующие действия:

1. Щелкните правой кнопкой мыши на папке макета и в контекстном меню выберите параметры New ▸ Other... (Создать ▸ Другой...) для отображения диалогового окна New (Создать).
2. В узле Android выберите параметр Android XML File (XML-файл Android) и щелкните на кнопке Next > (Далее >) для отображения диалогового окна New Android XML File (Создать XML-файл Android).
3. В текстовое поле File (Файл) введите имя guess_button.xml.
4. В разделе What type of resource would you like to create? (Тип создаваемого ресурса) выберите переключатель Layout (Макет). В результате новый файл guess_button.xml будет помещен в папку проекта res/layout.
5. В нижней части диалогового окна можно выбрать корневой элемент для нового макета. Выберите элемент Button.
6. Для завершения создания файла щелкните на кнопке Finish (Готово). Файл откроется в представлении XML.
7. Сконфигурируйте атрибуты компонента Button (листинг 6.5).

Листинг 6.5. Компонент newGuessButton может быть динамически «раздут» (guess_button.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<Button xmlns:android=http://schemas.android.com/apk/res/android
   android:id="@+id/newGuessButton" android:layout_weight="1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></Button>
```

6.4.6. Создание анимации «развевающегося флага»

С помощью XML-разметки, приведенной в листинге 6.6, определяется анимация «развевающегося флага», используемая в случае некорректно выбранного варианта. Способы применения этой анимации, определенной с помощью XML-разметки, описаны в разделе 6.5.

Листинг 6.6. Анимация «развевающегося флага» (incorrect_shake.xml), применяемая к флагу в случае некорректно выбранного варианта

```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <set xmlns:android=http://schemas.android.com/apk/res/android
4    android:interpolator="@android:anim/decelerate_interpolator">
5
6 <translate android:fromXDelta="0" android:toXDelta="-5%p"
7    android:duration="100"/>
8
9 <translate android:fromXDelta="-5%p" android:toXDelta="5%p"
10    android:duration="100" android:startOffset="100"/>
11
12 <translate android:fromXDelta="5%p" android:toXDelta="-5%p"
13    android:duration="100" android:startOffset="200"/>
14 </set>
```

Для создания этого файла анимации выполните следующие действия:

1. Щелкните правой кнопкой мыши на папке макета и в контекстном меню выберите параметры New ▸ Other... (Создать ▸ Другой...), чтобы отобразить диалоговое окно New (Создать).
2. В узле Android выберите параметр Android XML File (XML-файл Android) и щелкните на кнопке Next > (Далее >) для отображения диалогового окна New Android XML File (Создать XML-файл Android).
3. В текстовое поле File (Файл) введите имя incorrect_shake.xml.
4. В разделе What type of resource would you like to create? (Тип создаваемого ресурса) выберите переключатель Animation (Анимация). В результате новый файл incorrect_shake.xml будет помещен в папку проекта res/anim.
5. В нижней части диалогового окна можно установить корневой элемент анимации.
6. Для завершения создания файла щелкните на кнопке Finish (Готово). Файл откроется в представлении XML.
7. Сконфигурируйте анимацию (см. листинг 6.6).

В рассматриваемом примере используются анимации View для создания *эффекта «развевающегося флага»*. Эта анимация фактически состоит из трех анимаций, образующих *набор анимаций* (строки 3–14). Этот набор представляет собой коллекцию анимаций, образующих большую анимацию. Наборы анимаций могут включать произвольную комбинацию анимаций с переходами — alpha (прозрачность), scale (изменение размеров), translate (перемещение) и rotate. Используемая в приложении из этой главы анимация «развевающегося флага» включает набор из трех анимаций translate. Анимация translate перемещает компонент View внутри родительского компонента. Начиная с версии 3.0, Android поддерживает анимации свойств, позволяя задавать анимацию каждого свойства произвольного объекта. Анимации свойств будут использованы в приложении SpotOn Game в главе 8.

Первая анимация translate (строки 6–7) перемещает компонент View из начальной позиции в конечную через заданный период времени. Атрибут android:fromXDelta определяет смещение компонента View (в начальной позиции анимации), а атрибут

android:toXDelta — смещение компонента View в конечной позиции анимации. Эти атрибуты могут включать:

- абсолютные значения (в пикселях);
- процентные значения от размера анимированного компонента View;
- процентные значения от размера родительского компонента View.

Атрибуту android:fromXDelta было присвоено абсолютное значение 0. Атрибуту android:toXDelta было назначено значение -5%p, означающее, что компонент View должен быть перемещен влево (знак «минус») на 5 % от ширины родительского компонента (указано с помощью буквы p). Если нужно выполнить перемещение на величину, равную 5 % от ширины компонента View, букву p не указывайте. С помощью атрибута android:duration определяется длительность анимации (выражена в миллисекундах). Таким образом, анимация, определенная в строках 6–7, переместит компонент View влево на 5 % относительно ширины родительского компонента в течение 100 миллисекунд.

Вторая анимация (строки 9–10) продолжается с того места, где была завершена первая анимация. Она перемещает компонент View с позиции, заданной смещением -5%p, в позицию, заданную смещением %5p, в течение 100 миллисекунд. По умолчанию анимации, включенные в набор анимаций, применяются параллельно, но с помощью атрибута android:startOffset можно определить величину задержки (в миллисекундах) перед началом анимации. С помощью этой задержки реализуется последовательное выполнение анимаций. В рассматриваемом случае вторая анимация начинается через 100 миллисекунд после завершения первой анимации. Третья анимация (строки 12–13) совпадает со второй анимацией, но выполняется в обратном направлении и начинается через 200 миллисекунд после завершения первой анимации.

6.5. Создание приложения

В листингах 6.7–6.15 реализовано приложение Flag Quiz Game в единственном классе FlagQuizGame, расширяющем класс Activity.

Операторы package и import

В листинге 6.7 приведены операторы package и import класса FlagQuizGame.java. Оператор package в строке 3 указывает, что класс в этом файле входит в пакет com.deitel.flagquizgame (эта строка включается при создании проекта). В строках 5–35 импортируются различные классы Java и Android, а также интерфейсы, используемые приложением. Компоненты, которые используются исключительно при создании приложения в этой главе, были рассмотрены в разделе 6.3.

Листинг 6.7. Операторы package и import из приложения FlagQuizGames

```
1   // Класс FlagQuizGame.java
2   // Главный класс Activity для приложения Flag Quiz Game App
3
4   package com.deitel.flagquizgame;
5   import java.io.IOException;
6   import java.io.InputStream;
7   import java.util.ArrayList;
```

```
 8 import java.util.Collections;
 9 import java.util.HashMap;
10 import java.util.List;
11 import java.util.Map;
12 import java.util.Random;
13 import java.util.Set;
14
15 import android.app.Activity;
16 import android.app.AlertDialog;
17 import android.content.Context;
18 import android.content.DialogInterface;
19 import android.content.res.AssetManager;
20 import android.graphics.drawable.Drawable;
21 import android.os.Bundle;
22 import android.os.Handler;
23 import android.util.Log;
24 import android.view.LayoutInflater;
25 import android.view.Menu;
26 import android.view.MenuItem;
27 import android.view.View;
28 import android.view.View.OnClickListener;
29 import android.view.animation.Animation;
30 import android.view.animation.AnimationUtils;
31 import android.widget.Button;
32 import android.widget.ImageView;
33 import android.widget.TableLayout;
34 import android.widget.TableRow;
35 import android.widget.TextView;
36
```

Переменные экземпляра

В листинге 6.8 перечисляются переменные класса FlagQuizGame. В строке 40 объявляется переменная static final String TAG, используемая для регистрации сообщений об ошибках с помощью класса Log (см. листинг 6.10). Благодаря подобному приему разграничиваются сообщения об ошибках класса Activity от иных сообщений, записываемых в журнал устройства.

Объект List<String> fileNameList хранит названия файлов, в которых хранятся изображения флагов, для выбранных географических регионов. Объект List<String> quizCountriesList хранит 10 названий файлов флагов стран, используемых в вопросах викторины. Объект Map<String, Boolean> regionsMap хранит сведения, касающиеся используемых географических регионов.

Переменная String correctAnswer содержит имя файла флага для текущего правильного ответа относительно используемого флага. В переменной int totalGuesses хранится общее число правильных и неправильных вариантов ответа, выбранных пользователем. Переменная int correctAnswers хранить количество корректных вариантов, выбранных пользователем. Значение этой переменной станет равным 10 после завершения ответов пользователя на вопросы викторины. В переменной int guessRows хранится количество строк three-Button, отображающих варианты ответов на вопросы о флагах.

Объект Random random — это генератор псевдослучайных чисел, применяемый для случайного выбора флагов, которые будут включены в викторины, а также для случайного выбора строки и столбца, в которых будет помещена кнопка Button, соответствующая корректному ответу. С помощью объекта Handler handler реализуется задержка величиной в одну секунду для тестирования загрузки следующего флага.

Объект Animation shakeAnimation хранит динамически «раздуваемую» анимацию колышущегося флага, применяемую к изображению флага в случае некорректного выбора, сделанного пользователем. В строках 53–56 находятся переменные, применяемые для программного манипулирования различными компонентами GUI.

Листинг 6.8. Переменные экземпляра класса FlagQuizGame

```
37   public class FlagQuizGame extends Activity
38   {
39      // Строка, используемая при регистрации сообщений об ошибках
40      private static final String TAG = "FlagQuizGame Activity";
41
42      private List<String> fileNameList; // имена файлов флагов
43      private List<String> quizCountriesList;  // имена стран в викторине
44      private Map<String, Boolean> regionsMap; // используемые регионы
45      private String correctAnswer; // корректная страна текущего флага
46      private int totalGuesses; // количество сделанных предположений
47      private int correctAnswers; // число верных ответов
48      private int guessRows; // число строк, отображающих варианты ответов
49      private Random random; // генератор случайных чисел
50      private Handler handler; // задержка перед загрузкой
                                 // следующего флага
51      private Animation shakeAnimation; // анимация для неверного ответа
52
53      private TextView answerTextView; // отображение Correct!
                                         // или Incorrect!
54      private TextView questionNumberTextView; // # текущего вопроса
55      private ImageView flagImageView; // отображает флаг
56      private TableLayout buttonTableLayout; // таблица Buttons для ответа
57
```

Переопределение метода OnCreate для класса Activity

Метод onCreate (см. листинг 6.9) «раздувает» GUI и инициализирует переменные экземпляра класса Activity. Как и в случае с предыдущими приложениями, сначала вызывается метод onCreate из суперкласса (строка 62), а затем «раздувается» графический интерфейс пользователя для класса Activity (строка 63).

Листинг 6.9. Переопределение метода onCreate класса Activity

```
58   // вызывается при первом создании деятельности
59   @Override
60   public void onCreate(Bundle savedInstanceState)
61   {
62      super.onCreate(savedInstanceState); // вызов метода суперкласса
63      setContentView(R.layout.main);      // «раздувание» GUI
```

```
 64
 65    fileNameList = new ArrayList<String>(); // имена файлов изображений
 66    quizCountriesList = new ArrayList<String>(); // флаги для викторины
 67    regionsMap = new HashMap<String, Boolean>(); // Хеш-карта регионов
 68    guessRows = 1; // по умолчанию одна строка вариантов выбора
 69    random = new Random(); // инициализация генератора случайных чисел
 70    handler = new Handler(); // выполнение операций с задержкой
 71
 72    // загрузка анимации «развевающегося флага», используемой в случае
       // некорректных ответов
 73    shakeAnimation =
 74       AnimationUtils.loadAnimation(this, R.anim.incorrect_shake);
 75    shakeAnimation.setRepeatCount(3); // троекратное повторение анимации
 76
 77    // массив регионов мира из файла strings.xml
 78    String[] regionNames =
 79       getResources().getStringArray(R.array.regionsList);
 80
 81    // по умолчанию страны выбраны для всех регионов
 82    for (String region : regionNames)
 83    regionsMap.put(region, true);
 84
 85    // получение ссылок на компоненты GUI
 86    questionNumberTextView =
 87       (TextView) findViewById(R.id.questionNumberTextView);
 88    flagImageView = (ImageView) findViewById(R.id.flagImageView);
 89    buttonTableLayout =
 90       (TableLayout) findViewById(R.id.buttonTableLayout);
 91    answerTextView = (TextView) findViewById(R.id.answerTextView);
 92
 93    // настройка текста questionNumberTextView
 94    questionNumberTextView.setText(
 95       getResources().getString(R.string.question) + " 1 " +
 96       getResources().getString(R.string.of) + " 10");
 97
 98    resetQuiz(); // начало новой викторины
 99 } // завершение метода onCreate
100
```

В строках 65–66 создаются объекты ArrayList<String>, используемые для хранения названий файлов с изображениями флагов для выбранных географических регионов и десяти стран, отобранных для текущей викторины соответственно. В строке 67 создается объект HashMap<String, Boolean>, в котором хранятся сведения о том, выбран или нет тот или иной географический регион.

Переменной guessRows присвоено значение 1, вследствие чего игра изначально отображает лишь одну строку объектов Buttons, содержащую три возможных ответа. Пользователь может усложнить игру, отобразив две строки (с шестью возможными ответами) либо три строки (с девятью возможными ответами).

В строке 69 создается объект Random random, используемый для случайного выбора флагов, включаемых в викторину, а также для случайного выбора строки и столбца,

в которых будет находиться кнопка Button корректного ответа. В строке 70 создается объект Handler handler, применяемый для формирования задержки величиной в одну секунду перед отображением следующего флага в случае, если пользователь отгадал текущий флаг.

В строках 73–74 выполняется динамическая загрузка *анимации «развевающегося флага»*, применяемой к флагу в случае, если пользователь не угадал название страны, соответствующей этому флагу. Статический метод loadAnimation класса AnimationUtils загружает анимацию из XML-файла, представленного константой R.anim.incorrect_shake. Первый аргумент определяет компонент Context (экземпляр класса FlagQuizGame), включающий ресурсы, которые могут быть анимированы. В строке 75 определяется количество повторов анимации, реализуемых с помощью метода setRepeatCount класса Animation.

В строках 78–79 *динамически загружается* содержимое строкового массива regionNames. Метод getResources (косвенным образом наследуемый класса ContextWrapper) возвращает объект Resources (пакет android.content.res), который можно использовать для загрузки ресурсов класса Activity. Затем вызывается метод объекта getStringArray для загрузки массива, связанного с константой ресурса R.array.regionsList из файла strings.xml.

В строках 82–83 с помощью метода put каждый из шести регионов добавляется в число регионов HashMap. Каждому региону изначально присвоено значение true (используется). Пользователь может «включать» и «отключать» регионы с помощью меню параметров приложения (см. листинги 6.13–6.14).

В строках 86–91 выполняется получение ссылок на различные компоненты GUI, обрабатываемые программным способом. В строках 94–96 настраивается текст, отображаемый в questionNumberTextView. В данном случае используется строковое форматирование при создании текста questionNumberTextView. В разделе 7.4.3 мы рассмотрим создание строковых ресурсов для форматированных строк. В строке 98 вызывается метод resetQuiz класса FlagQuizGame, с помощью которого запускается следующая викторина.

Метод resetQuiz класса FlagQuizGame (наше приложение)

Метод resetQuiz (см. листинг 6.11) настраивает и запускает следующую викторину. Как упоминалось ранее, применяемые в игре изображения хранятся в папке приложения assets. Для получения доступа к содержимому папки метод использует возможности приложения AssetManager (строка 106). Для получения доступа к этому приложению вызывается метод getAssets (косвенно наследуется из класса ContextWrapper). Затем в строке 107 очищается список fileNameList, который таким образом подготавливается для загрузки имен файлов изображений, относящихся только к выбранным географическим регионам. Метод keySet (строка 111) класса HashMap применяется для формирования набора из шести названий регионов, взятых из regionsMap. Затем они присваиваются регионам, хранящимся в объекте Set<String>. После чего выполняется последовательный обход всех регионов (строки 114–124). Для каждого региона используется метод list из AssetManager (строка 119) для получения доступа к массиву, содержащему названия файлов изображений флагов, хранящиеся в строковом массиве paths. В строках 121–122 удаляется расширение .png из имени каждого файла изображения флага, а сами имена помещаются в список fileNameList.

Листинг 6.11. Метод resetQuiz класса FlagQuizGame

```
// настройка и запуск новой викторины
private void resetQuiz()
{
   // доступ к изображению флага с помощью AssetManager
   // имена файлов только для выбранных регионов
   AssetManager assets = getAssets(); // доступ AssetManager
                                      // для приложения
   fileNameList.clear(); // очистка списка

   try
   {
      Set<String> regions = regionsMap.keySet(); // получение
                                                 // набора регионов

      // циклический обход каждого региона
      for (String region : regions)
      {
         if (regionsMap.get(region)) // Активен ли регион
         {
            // список файлов с изображениями флагов для региона
            String[] paths = assets.list(region);

            for (String path : paths)
               fileNameList.add(path.replace(".png", ""));
        }   // конец блока if
    }   // конец блока for
}   // конец блока try
catch (IOException e)
{
    Log.e(TAG, "Error loading image file names", e);
}   // конец блока catch

correctAnswers = 0; // восстановление количества корректных вариантов
totalGuesses = 0;   // восстановление общего числа
                    //вариантов пользователя
quizCountriesList.clear();   // очистка предыдущего списка
                             // от стран викторины

// добавление 10 случайных имен файлов в список quizCountriesList
int flagCounter = 1;
int numberOfFlags = fileNameList.size(); // количество флагов

while (flagCounter <= 10)
{
   int randomIndex = random.nextInt(numberOfFlags); // случайный
                                                    // индекс

   // получение случайного имени файла
   String fileName = fileNameList.get(randomIndex);

```

продолжение ⇨

Листинг 6.11 *(продолжение)*

```
146       // если регион активен, но еще не выбран
147       if (!quizCountriesList.contains(fileName))
148       {
149          quizCountriesList.add(fileName); // добавление файла в список
150          ++flagCounter;
151       }  // конец блока if
152    }  // конец блока while
153
154       loadNextFlag(); // запуск викторины загрузкой первого флага
155    }  // конец метода resetQuiz
156
```

В строках 131–133 путем присваивания нулевых значений сбрасываются счетчики количества правильных вариантов ответов, выбранных пользователем (correctAnswers), и общего числа вариантов ответов, выбранных пользователем (totalGuesses), а также очищается список quizCountriesList.

В строках 136–152 добавляются 10 случайно выбранных имен файлов в список quizCountriesList. Изначально берется общее число флагов, затем случайным образом генерируется индекс в диапазоне от 0 до числа, на единицу меньшего, чем общее количество флагов. Затем с помощью индекса выбирается единственное название файла изображения в списке fileNamesList. Если в списке quizCountriesList не содержится имени файла, произойдет его добавление в этот список, а также увеличение на единицу счетчика flagCounter. Этот процесс повторяется до тех пор, пока не будут выбраны 10 уникальных названий файлов. Затем в строке 154 вызывается метод loadNextFlag (см. листинг 6.11) для загрузки первого флага викторины.

Методы loadNextFlag, getTableRow и getCountryName класса FlagQuizGame

Метод loadNextFlag (см. листинг 6.11) загружает и отображает следующий флаг вместе с соответствующим набором кнопок выбора вариантов ответов Buttons. Имена файлов изображений, хранящихся в списке quizCountriesList, записаны в формате, не использующем расширение .png:

```
regionName-countryName
```

Если regionName или countryName состоят из нескольких слов, для их разделения используется символ подчеркивания (_).

Листинг 6.11. Метод loadNextFlag класса FlagQuizGame

```
157    // после выбора пользователем корректного флага загружается
       // следующий флаг
158    private void loadNextFlag()
159    {
160       // получение имени файла для следующего флага и удаление
          // его из списка
161       String nextImageName = quizCountriesList.remove(0);
162       correctAnswer = nextImageName; // обновление корректного ответа
163
```

```
      answerTextView.setText(""); // очистка answerTextView

      // отображение номера текущего вопроса викторины
      questionNumberTextView.setText(
         getResources().getString(R.string.question) + " " +
         (correctAnswers + 1) + " " +
         getResources().getString(R.string.of) + " 10");

      // выборка региона из имени следующего изображения
      String region =
         nextImageName.substring(0, nextImageName.indexOf('-'));

      // использование AssetManager для загрузки следующего
      // изображения из папки assets
      AssetManager assets = getAssets(); // получение AssetManager
                                         // приложения
      InputStream stream; // используется для чтения изображений флагов

      try
      {
         // получение InputStream ресурса, представляющего следующий флаг
         stream = assets.open(region + "/" + nextImageName + ".png");

         // загрузка ресурса в качестве Drawable и отображение
         // в flagImageView
         Drawable flag = Drawable.createFromStream
            (stream, nextImageName);
        flagImageView.setImageDrawable(flag);
     }  // конец блока try
     catch (IOException e)
    {
       Log.e(TAG, "Error loading " + nextImageName, e);
  } // конец блока catch

    // очистка предыдущих кнопок ответа Buttons из TableRows
    for (int row = 0; row < buttonTableLayout.getChildCount(); ++row)
       ((TableRow) buttonTableLayout.getChildAt(row)).removeAllViews();

    Collections.shuffle(fileNameList); // имена «перемешанных» файлов

    // помещение корректного ответа в конец списка fileNameList
    int correct = fileNameList.indexOf(correctAnswer);
    fileNameList.add(fileNameList.remove(correct));

    // получение ссылки на службу LayoutInflater
    LayoutInflater inflater = (LayoutInflater) getSystemService(
       Context.LAYOUT_INFLATER_SERVICE);

    // добавление 3, 6 и 9 кнопок ответов Buttons на основе
    // значения guessRows
    for (int row = 0; row < guessRows; row++)
```

продолжение ⇗

Листинг 6.11 *(продолжение)*

```
210       {
211          TableRow currentTableRow = getTableRow(row);
212
213          // помещение кнопок Buttons в currentTableRow
214          for (int column = 0; column < 3; column++)
215          {
216          // «раздувание» guess_button.xml для создания новой кнопки Button
217             Button newGuessButton =
218                (Button) inflater.inflate(R.layout.guess_button, null);
219
220             // получение названия страны и использование в виде
                // текста newGuessButton
221             String fileName = fileNameList.get((row * 3) + column);
222             newGuessButton.setText(getCountryName(fileName));
223
224             // регистрация answerButtonListener для ответа
                // на нажатия кнопок
225             newGuessButton.setOnClickListener(guessButtonListener);
226             currentTableRow.addView(newGuessButton);
227          }  // конец цикла for
228       }  // конец цикла for
229
230       // случайная замена одной кнопки Button корректным ответом
231       int row = random.nextInt(guessRows); // выбор случайной строки
232       int column = random.nextInt(3); // выбор случайного столбца
233       TableRow randomTableRow = getTableRow(row); // получение TableRow
234       String countryName = getCountryName(correctAnswer);
235       ((Button)randomTableRow.getChildAt(column)).setText(countryName);
236    } // конец метода loadNextFlag
237
238    // возвращение указанного компонента TableRow
239    private TableRow getTableRow(int row)
240    {
241       return (TableRow) buttonTableLayout.getChildAt(row);
242    }  // конец метода getTableRow
243
244    // разбор имени файла флага и возвращение названия страны
245    private String getCountryName(String name)
246    {
247       return name.substring(name.indexOf('-') + 1).replace('_', ' ');
248    }  // конец метода getCountryName
249
```

В строке 161 происходит удаление первого имени из quizCountriesList, которое при этом сохраняется в nextImageName. Это имя также сохраняется в переменной correctAnswer и в дальнейшем может использоваться для определения корректности выбранного пользователем варианта ответа. Затем переменная answerTextView очищается и номер текущего ответа отображается с помощью компонента questionNumerTextView

(строки 164–170). И снова можно воспользоваться ресурсом в виде форматированной строки, как указано в главе 7.

В строках 173–174 из nextImageName извлекается регион, который будет использоваться в качестве имени подпапки в папке assets, содержащей изображение. Затем запускается AssetManager, используемый в конструкции try для открытия потока InputStream, с помощью которого осуществляется чтение из файла, содержащего изображение флага. Этот поток используется в качестве аргумента статического метода createFromStream класса Drawable, создающего объект Drawable.

Объект Drawable используется в качестве элемента компонента flagImageView для формирования отображения с помощью метода setImageDrawable. Если в блоке try возникает исключение (строки 180–188), оно фиксируется с помощью встроенного механизма регистрации Android (в целях отладки). Этот механизм поддерживает статические методы, которые обеспечивают различные аспекты регистрации сообщений. Статический метод регистрации e применяется для регистрации ошибок (как минимум, в терминах сгенерированных сообщений об ошибках). Для получения дополнительных сведений о регистрации ошибок обратитесь к полному списку методов регистрации, приведенному на веб-сайте developer.android.com/reference/android/util/Log.html.

В строках 195–196 удаляются все предыдущие кнопки ответов Buttons из трех компонентов TableRows макета buttonTableLayout. Затем в строке 198 «перемешивается» содержимое списка fileNameList, а в строках 201–202 осуществляется поиск ответа correctAnswer и его перемещение в конец списка fileNameList. После чего этот ответ случайным образом включается в кнопку ответа answer Buttons.

В строках 205–206 реализуется доступ к объекту LayoutInflater, необходимому для «раздувания» объекта Button, используемого для ответов на вопросы, из файла разметки guess_button.xml. В строках 209–228 выполняется итерация по строкам и столбцам buttonTableLayout (в соответствии с текущим числом guessRows). Для каждого нового объекта Button выполняются следующие действия:

- в строках 217–218 «раздувается» объект Button из файла guess_button.xml;
- в строке 221 выбирается имя файла флага;
- в строке 222 в качестве текста компонента Button выбирается название страны;
- в строке 225 настраивается OnClickListener для нового объекта Button;
- в строке 226 новая кнопка Button добавляется в соответствующий компонент TableRow.

В строках 231–235 выбирается случайная строка (на основе текущего номера guessRows) и столбец компонента buttonTableLayout, а затем в качестве текста кнопки Button, находящейся в соответствующей строке и столбце, присваивается корректный ответ.

В строках 211 и 233 в методе loadNextFlag используется метод утилиты getTableRow (строки 239–242) для получения TableRow со специфическим индексом в buttonTableLayout. В строках 222 и 234 используется метод утилиты getCountryName (строки 245–248) для разбора имени страны на основе имени файла изображения.

Методы submitGuess и disableButtons класса FlagQuizGame

Метод submitGuess (см. листинг 6.12) вызывается после щелчка пользователя на кнопке Button, соответствующей стране, для выбора варианта ответа. В качестве параметра guessButton метод принимает кнопку Button, на которой был произведен щелчок мышью. Мы получили текст кнопки Button (строка 253) и проанализированное название страны (строка 254), затем выполняется приращение значения переменной totalGuesses.

Листинг 6.12. Метод submitGuess класса FlagQuizGame

```
250 // вызывается, если пользователь выберет вариант ответа
251 private void submitGuess(Button guessButton)
252 {
253    String guess = guessButton.getText().toString();
254    String answer = getCountryName(correctAnswer);
255    ++totalGuesses; // увеличение на единицу номера ответа пользователя
256
257    // если ответ пользователя корректный
258    if (guess.equals(answer))
259    {
260       ++correctAnswers; // увеличение на 1 количества верных ответов
261
262       // отображение зеленым цветом слова "Correct!"
263       answerTextView.setText(answer + "!");
264       answerTextView.setTextColor(
265          getResources().getColor(R.color.correct_answer));
266
267       disableButtons(); // отключение всех кнопок ответа Buttons
268
269       // если пользователь корректно идентифицировал 10 флагов
270       if (correctAnswers == 10)
271       {
272          // создание нового AlertDialog Builder
273          AlertDialog.Builder builder = new AlertDialog.Builder(this);
274
275          builder.setTitle(R.string.reset_quiz); // строка заголовка
276
277          // создание сообщения AlertDialog, отображающего
             // результаты игры
278          builder.setMessage(String.format("%d %s, %.02f%% %s",
279             totalGuesses, getResources().getString(R.string.guesses),
280                (1000 / (double) totalGuesses),
281                getResources().getString(R.string.correct)));
282
283             builder.setCancelable(false);
284
285             // добавление кнопки Button "Reset Quiz"
286             builder.setPositiveButton(R.string.reset_quiz,
287                new DialogInterface.OnClickListener()
288                {
289                   public void onClick(DialogInterface dialog, int id)
290                   {
```

```
291                         resetQuiz();
292                      }  // конец метода onClick
293                  }   // конец анонимного внутреннего класса
294             };   // завершение вызова setPositiveButton
295
296             // создание AlertDialog на основе Builder
297             AlertDialog resetDialog = builder.create();
298             resetDialog.show(); // отображение диалогового окна
299         }   // завершение блока if
300         else // ответ корректен, но викторина еще не завершена
301         {
302            // загрузка следующего флага через одну секунду
303            handler.postDelayed(
304            new Runnable()
305            {
306               @Override
307               public void run()
308               {
309                  loadNextFlag();
310               }
311            }, 1000); // 1000 миллисекунд (секундная задержка)
312         } // конец else
313     }   // конец if
314     else // выбран некорректный вариант
315     {
316        // воспроизведение анимации
317        flagImageView.startAnimation(shakeAnimation);
318
319        // отображение красным цветом слова "Incorrect!"
320        answerTextView.setText(R.string.incorrect_answer);
321        answerTextView.setTextColor(
322           getResources().getColor(R.color.incorrect_answer)
323           guessButton.setEnabled(false); // отключение
                                              // некорректного ответа
324      }   // конец else
325  }   // конец метода submitGuess
326
327  // метод утилиты, отключающий все кнопки Buttons ответа
328  private void disableButtons()
329  {
330     for (int row = 0; row < buttonTableLayout.getChildCount(); ++row)
331     {
332        TableRow tableRow = (TableRow) buttonTableLayout.getChildAt(row);
333        for (int i = 0; i < tableRow.getChildCount(); ++i)
334            tableRow.getChildAt(i).setEnabled(false);
335        }   // конец внешнего цикла for
336     }   // конец метода disableButtons
337
```

Если предложенный пользователем вариант ответа (строка 258) корректен, выполняется увеличение на единицу значение счетчика correctAnswers. Затем в качестве текста

компонента answerTextView выбирается название страны, а также цвет, представленный константой R.color.correct_answer. Затем вызывается метод утилиты disableButtons (определен в сроках 328–336), с помощью которого выполняется обход строк и столбцов компонента buttonTableLayout и отключение всех кнопок ответа Buttons.

Если значение переменной correctAnswers становится равным 10 (строка 270), викторина завершается. В строках 273–299 создается новый AlertDialog.Builder, который используется для конфигурирования диалогового окна, отображающего результаты викторины. Также создается диалоговое окно AlertDialog, отображаемое на экране. Как только пользователь коснется кнопки Reset Quiz, вызывается метод resetQuiz, который начинает новую игру.

Если значение переменной correctAnswers меньше 10, в строках 303–311 вызывается метод postDelayed объекта Handler handler. Первый аргумент этого метода определяет анонимный внутренний класс, который реализует интерфейс Runnable. С помощью этого интерфейса можно задать задержку в несколько миллисекунд (loadNextFlag), по истечении которой будет выполнена запланированная задача. В качестве второго аргумента используется задержка, выраженная в миллисекундах (1000).

Если выбранный пользователем вариант некорректен, в строке 317 вызывается метод startAnimation класса flagImageView, с помощью которого воспроизводится анимация shakeAnimation, которая загружается в метод onCreate. Также в качестве текста answerTextView выбирается слово «Incorrect!», выделенное красным цветом (строки 320–322), затем вызывается метод setEnabled класса guessButton со значением false (строка 323), с помощью которого *деактивируется* кнопка Button, которая соответствует некорректному ответу.

Переопределение метода onCreateOptionsMenu класса Activity

Мы переопределяем метод OnCreateOptionsMenu класса Activity (см. листинг 6.13) для инициализации меню стандартных параметров класса Activity. Система передает объект Menu, в котором отображаются параметры. Приложение имеет свой собственный набор встроенных параметров, с помощью которых пользователь может выбирать одно из двух меню — Select Number of Choices (Выбрать число вариантов) или Select Regions (Выбрать регионы). Параметр Select Number of Choices позволяет пользователю выбрать количество флагов 3, 6 или 9, отображаемых для каждой викторины. С помощью параметра Select Regions пользователь может выбрать географический регион, из которого будут выбираться страны для викторины.

Листинг 6.13. Переопределение метода onCreateOptionsMenu класса Activity

```
// создание констант для каждого идентификатора меню
private final int CHOICES_MENU_ID = Menu.FIRST;
private final int REGIONS_MENU_ID = Menu.FIRST + 1;

// вызывается, если пользователь получает доступ к параметрам меню
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
   super.onCreateOptionsMenu(menu);

```

```
348       // добавляет два параметра в меню — "Choices" и "Regions"
349       menu.add(Menu.NONE, CHOICES_MENU_ID, Menu.NONE, R.string.choices);
350       menu.add(Menu.NONE, REGIONS_MENU_ID, Menu.NONE, R.string.regions);
351
352       return true; // отображение меню
353    }  // конец метода onCreateOptionsMenu
354
```

В строках 339–340 создаются константы для двух ID меню. Константа Menu.FIRST представляет параметр, который будет сначала появляться в меню Menu. Каждый параметр имеет уникальный ID. Метод onCreateOptionsMenu сначала вызывает метод onCreateOptionsMenu суперкласса. Затем вызывается метод add из Menu, добавляющий MenuItems в Menu (строки 349–350). Первый аргумент представляет ID группы MenuItem для группирования MenuItems, которые имеют одинаковое состояние (например, являются активированными или видимыми на экране). В качестве аргумента может использоваться Menu.NONE, если компонент MenuItem не является частью группы. Второй аргумент — уникальный элемент ID MenuItem. Третий аргумент определяет порядок, в котором отображаются MenuItem. Используйте Menu.NONE, если порядок следования MenuItems не имеет значения. Последний аргумент является идентификатором ресурса String, который будет отображаться. Для отображения меню возвращается значение true (строка 352).

Переопределение метода onOptionsItemSelected класса Activity

Метод onOptionsItemSelected (см. листинг 6.14) вызывается в случае, если пользователь выбирает элемент в меню параметров приложения и получает выбранный элемент MenuItem. Оператор switch применяется для выбора между двумя случаями. Управляющее выражение switch вызывает метод getItemId компонента item, возвращающий уникальный идентификатор элемента меню (строка 360), благодаря которому можно определить, какой именно элемент MenuItem был выбран.

Листинг 6.14. Переопределение метода onOptionsItemSelected класса Activity

```
355    // вызывается, если пользователь выбирает элемент меню
356    @Override
357    public boolean onOptionsItemSelected(MenuItem item)
358    {
359       // выбирается идентификатор меню, соответствующий
          // выбранному пользователем параметру меню
360       switch (item.getItemId())
361       {
362           case CHOICES_MENU_ID:
363              // создание списка возможных номеров вариантов ответов
364              final String[] possibleChoices =
365                 getResources().getStringArray(R.array.guessesList);
366
367              // создание нового AlertDialog Builder и выбор заголовка
368              AlertDialog.Builder choicesBuilder =
369                 new AlertDialog.Builder(this);
370              choicesBuilder.setTitle(R.string.choices);
```

продолжение ⇨

Листинг 6.11 *(продолжение)*

```

            // добавление элементов possibleChoices в диалоговое окно
            // и настройка поведения элементов, на которые произведен
            // щелчок
            choicesBuilder.setItems(R.array.guessesList,
               new DialogInterface.OnClickListener()
               {
                  public void onClick(DialogInterface dialog, int item)
                  {
                     // обновление guessRows, чтобы соответствовать
                     // выбору пользователя
                     guessRows = Integer.parseInt(
                        possibleChoices[item].toString()) / 3;
                     resetQuiz(); // переустановка викторины
                  }  // конец метода onClick
               }  // конец анонимного внутреннего класса
            ); // завершение вызова setItems

            // создание диалогового окна AlertDialog на основе Builder
            AlertDialog choicesDialog = choicesBuilder.create();
            choicesDialog.show(); // отображение диалогового окна
            return true;

         case REGIONS_MENU_ID:
         // получение массива регионов мира
         final String[] regionNames =
            regionsMap.keySet().toArray(new String[regionsMap.size()]);

         // булевский массив, представляющий выбор регионов
         boolean[] regionsEnabled = new boolean[regionsMap.size()];
         for (int i = 0; i < regionsEnabled.length; ++i)
            regionsEnabled[i] = regionsMap.get(regionNames[i]);

         // создание AlertDialog Builder и выбор заголовка
         // диалогового окна
         AlertDialog.Builder regionsBuilder =
            new AlertDialog.Builder(this);
         regionsBuilder.setTitle(R.string.regions);

         // замена _ пробелом в именах регионов перед отображением
         String[] displayNames = new String[regionNames.length];
         for (int i = 0; i < regionNames.length; ++i)
            displayNames[i] = regionNames[i].replace('_', ' ');

         // добавление displayNames в диалоговое окно
         // и настройка поведения после щелчка на элементах
         regionsBuilder.setMultiChoiceItems(
            displayNames, regionsEnabled,
            new DialogInterface.OnMultiChoiceClickListener()
            {
```

```
418             @Override
419             public void onClick(DialogInterface dialog, int which,
420                boolean isChecked)
421                {
422                   // включение/исключение «щелкнутого» региона
423                   // в зависимости от того, выбран он или нет
424                   regionsMap.put(
425                      regionNames[which].toString(), isChecked);
426                }  // конец метода onClick
427             }  // конец анонимного внутреннего класса
428          );  // завершение вызова setMultiChoiceItems
429
430          // реинициализация викторины после нажатия кнопки "Reset Quiz"
431          regionsBuilder.setPositiveButton(R.string.reset_quiz,
432             new DialogInterface.OnClickListener()
433             {
434                @Override
435                public void onClick(DialogInterface dialog, int button)
436                {
437                   resetQuiz(); // переустановка викторины
438                }  // конец метода onClick
439             }  // конец анонимного внутреннего класса
440       );  // завершение вызова метода setPositiveButton
441
442       // создание диалогового окна на основе Builder
443       AlertDialog regionsDialog = regionsBuilder.create();
444       regionsDialog.show(); // отображение диалогового окна
445       return true;
446   }  // конец блока switch
447
448       return super.onOptionsItemSelected(item);
449    }  // конец метода onOptionsItemSelected
450
```

Если пользователь коснется параметра Select Number of Choices, вызывается вариант выбора, определенный в строках 362–390. В строках 364–365 из ресурсов приложения извлекается строковый массив guessesList и присваивается переменной possibleChoices. Затем создается новое диалоговое окно AlertDialog.Builder, которому присваивается заголовок (строки 368–370).

Каждое из ранее созданных диалоговых окон AlertDialogs отображает простое текстовое сообщение, а также одну или две кнопки Buttons. В этом случае мы отображаем элементы possibleChoice в диалоговом окне и указываем, что произойдет в том случае, если пользователь коснется одного из элементов. Чтобы выполнить все это, вызовем метод setItems класса AlertDialog.Builder (строки 374–385). В качестве первого аргумента используется массив строк или константа ресурса, которая представляет массив строк (представляет набор взаимно исключающих параметров). Второй аргумент — DialogInterface.OnClickListener — отвечает на касание пользователем одного из элементов. Метод onClick класса listener получает в качестве второго аргумента начинающийся с нуля индекс элемента, которого касается пользователь. Этот индекс применяется для выбора соответствующего элемента из possibleChoices, затем выполняется

преобразование типа String в int, после чего результат делится на 3 для определения количества guessRows. Затем вызывается метод resetQuiz для запуска новой викторины с указанным количеством кнопок ответа Buttons. В строках 388–389 создается и отображается диалоговое окно.

Если пользователь коснется параметра Select Regions, вызывается вариант выбора, определенный в строках 392–445, отображающий диалоговое окно AlertDialog, включающее перечень названий регионов, в которых выбираются несколько элементов. Во-первых, мы назначаем regionNames массив строк, содержащих ключи из regionsMap (строки 394–395). А затем в строках 398–400 создается массив типа boolean, в котором определяется выбор того или иного региона. В строках 403–405 создается диалоговое окно AlertDialog.Builder и присваивается имя диалоговому окну. В строках 408–410 создается массив строк displayNames, содержащий названия регионов, в которых символы подчеркивания заменены пробелами.

Затем вызывается метод setMultiChoiceItems класса AlertDialog.Builder, выполняющий отображение списка регионов. Для каждого выбранного региона устанавливается флажок (см. рис. 6.6). Первые два аргумента представляют собой массивы элементов, один из которых применяется для отображения элементов, а второй является массивом типа boolean, с помощью которого определяются используемые элементы. В качестве первого элемента может использоваться массив строк или константа ресурса, представляющая массив строк. В качестве третьего аргумента применяется интерфейс DialogInterface.OnMultiChoiceClickListener, который «отвечает» на каждое касание элемента в диалоговом окне. Анонимный внутренний класс (строки 416–427) реализует метод onClick слушателя (listener), применяемый для включения или исключения региона, на котором выполнен щелчок пользователя. Второй аргумент метода представляет индекс элемента, которого может коснуться пользователь. Третий аргумент представляет «выбранное» состояние компонента. Этот аргумент применяется для передачи информации об обновленном состоянии компонента в regionsMap.

В строках 431–440 определяется «положительная» кнопка Button диалогового окна. Если пользователь касается этой кнопки, вызывается метод resetQuiz, используемый для запуска новой игры, основанной на текущих настройках игры. Если пользователь просто коснется кнопки Back (Назад) устройства, новые настройки не будут использоваться до тех пор, пока не начнется новая викторина. И наконец, в строках 443–444 создается и отображается диалоговое окно.

Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события, порождаемые кнопками Buttons выбора вариантов ответа пользователем

Анонимный внутренний объект класса guessButtonListener реализует интерфейс OnClickListener, который «отвечает» на события, порождаемые кнопкой Button. В строке 225 регистрируется guessButtonListener в виде объекта-обработчика событий для каждого нового newGuessButton. Метод onClick просто передает кнопку Button методу submitGuess (см. листинг 6.15).

Листинг 6.15. Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события answerButton

```
// вызывается, если нажимается кнопка варианта ответа
private OnClickListener guessButtonListener = new OnClickListener()
{
   @Override
   public void onClick(View v)
   {
      submitGuess((Button) v); // передается выделенная кнопка
                               // Button в submitGuess
   }  // завершение метода onClick
}; // завершение метода answerButtonListener
} // завершение класса FlagQuizGame
```

6.6. Файл AndroidManifest.xml

В разделе 5.6 вы впервые познакомились с содержимым файла манифеста. Для приложения, разрабатываемого в этой главе, рассматриваются только новые функции и свойства файла манифеста (листинг 6.16). В строке 7 используется атрибут android:theme элемента application для применения темы к графическому интерфейсу приложения. Тема представляет собой набор стилей, с помощью которых определяется отображение компонентов GUI. В рассматриваемом случае значение атрибута определяет скрытие заголовка приложения, в котором отображается имя приложения. Полный список предопределенных стилей и тем вы найдете на веб-сайте developer.android.com/reference/android/R.style.html.

Дополнительные сведения о применении стилей и тем вы найдете на веб-сайте developer.android.com/guide/topics/ui/themes.html.

Чтобы выбрать тему приложения, воспользуйтесь вкладкой Application (Приложение) редактора манифеста. Введите значение атрибута, показанное в строке 7, в поле Theme (Тема).

В элементе activity (строка 10) используется атрибут android:screenOrientation для определения отображения информации приложением в *портретном режиме* (вертикальная ориентация). Чтобы установить значение атрибута, выберите деятельность в левом нижнем углу вкладки Application в окне редактора манифеста. Параметры манифеста для деятельности отображаются в правом нижнем углу вкладки Application. В раскрывающемся списке Screen orientation (Ориентация экрана) выберите параметр portrait (портретная). После внесения изменений в манифест не забудьте сохранить изменения.

Листинг 6.16. Файл AndroidManifest.xml для приложения Flag Quiz Game

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android=http://schemas.android.com/apk/res/android
3     package="com.deitel.flagquizgame" android:versionCode="1"
4     android:versionName="1.0">
5     <application android:icon="@drawable/icon"
6        android:label="@string/app_name"
7        android:theme="@android:style/Theme.NoTitleBar">
```

продолжение ⇗

Листинг 6.16 *(продолжение)*

```
     <activity android:name=".FlagQuizGame"
        android:label="@string/app_name"
        android:screenOrientation="portrait">
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
     </activity>
  </application>
  <uses-sdk android:targetSdkVersion="10" android:minSdkVersion="8"/>
</manifest>
```

6.7. Резюме

В этой главе было создано приложение Flag Quiz Game, используемое для проверки знания пользователем флагов различных государств. Был изучен порядок определения массивов String в файле strings.xml. Вы освоили порядок загрузки ресурсов цвета и массивов String из файлов colors.xml и strings.xml в память с помощью объекта Resources класса Activity.

Для отображения приложением флага, используемого в вопросе викторины, применялся диспетчер AssetManager. С помощью этого диспетчера открывался поток, в который считывался файл изображения флага. Затем этот поток использовался вместе со статическим методом createFromStream класса Drawable для создания объекта Drawable, который может отображаться в компоненте ImageView с помощью метода setImageDrawable компонента ImageView.

Вы изучили, каким образом с помощью компонента Menu приложения можно разрешить пользователю сконфигурировать параметры приложения. Чтобы определить параметры Menu, использовалось переопределение метода onCreateOptionsMenu класса Activity. Чтобы «ответить» на выбор элементов меню пользователем, выполнялось переопределение метода onOptionsItemSelected Activity.

Для определения задержки перед отображением следующего флага после выбора пользователем корректного варианта использовался метод postDelayed объекта Handler для запуска на выполнение Runnable после задержки, равной 1000 миллисекунд. Если пользователь сделает неправильный выбор, приложение «встряхивает» флаг путем применения анимации к компоненту ImageView. Был использован статический метод loadAnimation класса AnimationUtils для загрузки анимации из XML-файла, определяющего параметры анимации. Также было определено количество повторов анимации с помощью метода setRepeatCount класса Animation, а сама анимация была реализована путем вызова метода startAnimation (причем в качестве аргумента использовался класс Animation) класса View для компонента ImageView.

Была изучена методика регистрации исключений (в целях отладки) с помощью встроенного механизма регистрации исключений Android. Этот механизм использует циркулярный буфер для кратковременного хранения сообщений. Для управления

данными в приложении также использовались различные классы коллекций и интерфейсы из пакета java.util.

В главе 7 мы разработаем приложение Cannon Game. При его создании будет использована многопоточность и покадровая анимация, выполняться обработка жестов, а также применен таймер для генерирования событий и обновления изображения на экране в ответ на эти события. Также мы рассмотрим простую методику идентификации коллизий.

7

Приложение Cannon Game

Прослушивание касаний и жестов, покадровая анимация, графика, звук, потоки, SurfaceView и SurfaceHolder

В этой главе...

- Создание игры, простой в кодировании и удивительно интересной.
- Создание подкласса SurfaceView и его использование для отображения игровой графики в отдельном потоке выполнения.
- Создание графики с помощью Paints и Canvas.
- Переопределение метода onTouchEvent класса Activity для обработки событий, связанных с касаниями, после прикосновений пользователя к экрану либо перетаскиванием пальцем в области экрана.
- Использование GestureDetector для распознавания более сложных «касательных» движений пользователя, например двойных касаний.
- Обнаружение простых столкновений.
- Добавление звука в приложение с помощью SoundPooll и AudioManager.
- Переопределение трех дополнительных методов «жизненного цикла» класса Activity.

7.1. Введение

В игре Cannon Game требуется разрушить состоящую из семи частей мишень в течение 10 секунд, отведенных на игру (рис. 7.1). Игра включает четыре визуальных компонента — *пушка*, управляемая пользователем, *пушечное ядро*, *мишень* и *блок*, защищающий мишень. Чтобы нацелить пушку, коснитесь пальцем экрана. В результате пушка

нацелится в точку, в которой вы коснулись экрана пальцем. После двойного касания пальцем экрана пушка выстрелит пушечным ядром. В конце игры приложение отобразит диалоговое окно AlertDialog, с помощью которого можно узнать, выиграли вы или проиграли, количество сделанных выстрелов и время игры (рис. 7.2).

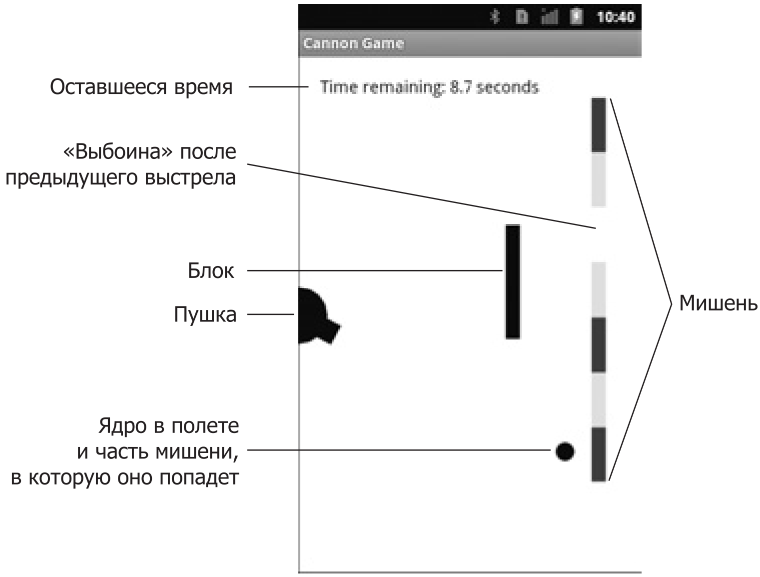


Рис. 7.1. Завершенное приложение Cannon Game

В начале игры каждому пользователю выделяется *лимит времени, равный 10 секундам*. При каждом попадании в секцию мишени к лимиту времени *добавляются* три секунды, а при каждом попадании в блок *вычитаются* две секунды. Игра считается выигранной, если были разрушены все секции мишени до истечения заданного лимита времени. Если значение таймера становится равным нулю до разрушения мишени, вы проиграете.

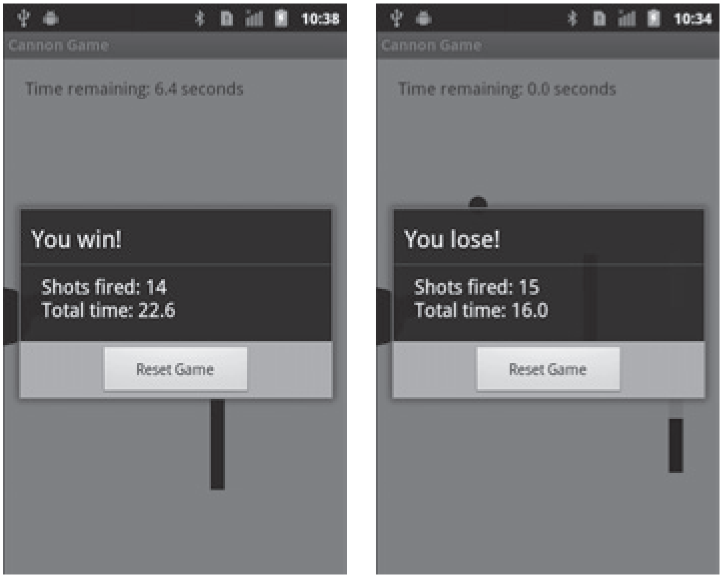
После выстрела из пушки приложение воспроизводит *звук выстрела*. Мишень состоит из семи секций. После попадания пушечного ядра в секцию мишени раздается *звук разбивающегося стекла*, а сама секция исчезает с экрана. Если ядро попадает в блок, раздается *звук удара*, а ядро отскакивает назад. При этом блок не разрушается. Мишень и блок перемещаются *по вертикали* с различной скоростью. Причем направление перемещения изменяется после касания верхней или нижней части экрана.

7.2. Тестирование приложения Cannon Game

Открытие и выполнение приложения

Откройте среду Eclipse и импортируйте проект Cannon Game app. Выполните следующие действия:

1. *Отройте диалоговое окно* Import. Для этого выполните команды File ▸ Import... (Файл ▸ Импорт...).



а *б*

Рис. 7.2. С помощью диалогового окна AlertDialogs приложения Cannon Game можно узнать, выиграли вы или проиграли: *а* — после разрушения всех семи секций мишени соответствующее сообщение появляется в окне AlertDialog; *б* — если время истечет до разрушения всех секций мишени, в окне AlertDialog отобразится соответствующее сообщение

2. *Импортируйте проект приложения* Canon Game. В диалоговом окне Import раскройте узел General (Общий) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область). Затем щелкните на кнопке Next > (Далее >) для выполнения шага Import Projects (Импорт проектов). Выберите корневой каталог и щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папки) выберите папку CanonGame в папке примеров книги и щелкните на кнопке OK. Щелкните на кнопке Finish (Готово) для импорта проекта в среду Eclipse. Проект отобразится в окне Package Explorer, находящемся в левой части окна Eclipse.
3. *Запустите приложение* Canon Game. В среде Eclipse щелкните правой кнопкой мыши на проекте CanonGame, находящемся в окне Package Explorer, затем в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android).

Процесс игры

Перетащите палец в области экрана или коснитесь им экрана, чтобы нацелить пушку. Дважды коснитесь экрана, чтобы выстрелить из пушки. Вы сможете выстрелить пушечным ядром только в том случае, если на экране не отображается другое ядро. Если приложение выполняется на экране AVD, в качестве «пальца» будет использоваться

мышь. Попытайтесь уничтожить мишень как можно быстрее. Если показания таймера обнулятся, игра завершится.

7.3. Обзор применяемых технологий

В этом разделе вашему вниманию представлены несколько новых технологий, использованных при создании приложения Cannon Game. Эти технологии будут рассмотрены в том порядке, в котором они встречаются в главе.

Определение ресурсов форматированных строк в файле strings.xml

В этом приложении определяются ресурсы String, используемые для представления форматированных строк. Эти строки применяются при вызовах метода getString (или статического метода format класса String) класса Resource. Если форматные строки включают несколько спецификаторов формата, потребуется пронумеровать их (начиная с 1) для определения порядка подстановки соответствующих значений в форматированную строку. Фразы на различных языках могут представлять собой результат подстановки значений в различные места локализованных строковых ресурсов. В подобных случаях локализованные версии файла strings.xml используют оригинальные номера спецификаторов формата, которые помещаются в соответствующие места локализованных строк. Синтаксис, используемый в формате спецификаторов нумерации, будет рассмотрен в разделе 7.4.3.

Добавление в макет пользовательского представления

Можно создать пользовательское представление путем расширения класса View или одного из его подклассов. Пример подобного представления — класс CannonView (раздел 7.5.3), который расширяет класс SurfaceView (будет рассмотрен в ближайшее время). Чтобы добавить пользовательский компонент в файл XML-разметки, следует полностью определить имя соответствующего ему класса в XML-элементе, представляющем компонент. Эта методика рассматривается в разделе 7.4.4.

Использование папки ресурсов raw

Медиафайлы (например, звуки), используемые в приложении **Cannon Game**, находятся в папке ресурсов приложения res/raw. Создание этой папки рассматривается в разделе 7.4.5. В эту папку помещаются звуковые файлы.

Методы «жизненного цикла» onPause и onDestroy класса Activity

Это приложение использует дополнительные методы «жизненного цикла» Activity. Метод onPause вызывается для текущего класса Activity в случае, если другая деятельность получает фокус. При этом текущая деятельность будет выполняться в фоновом режиме. С помощью метода onPause текущая игра приостанавливается. То есть если пользователь не взаимодействует с игрой, выполнение игры прекращается.

Если класс Activity завершается, вызывается его метод onDestroy. Этот метод применяется для «освобождения» звуковых ресурсов приложения. Методы «жизненного цикла» используются в разделе 7.5.2.

Переопределение метода onTouchEvent класса Activity

Как уже упоминалось ранее, пользователи взаимодействуют с приложением путем касания экрана устройства. Чтобы нацелить пушку на определенную точку экрана, *коснитесь* (*или один раз нажмите*) пальцем. Для выполнения обработки событий класса Activity, вызываемых касанием экрана пальцем, переопределите onTouchEvent класса Activity (раздел 7.5.2), а затем воспользуйтесь константами класса MotionEvent (пакет android.view) для определения типа происходящих событий и их последующей обработки.

Классы GestureDetector и SimpleOnGestureListener

Для обработки более сложных жестов, например *двойных* нажатий, используемых для выполнения стрельбы из пушки, можно воспользоваться классом GestureDetector (пакет android.view). С помощью этого класса распознаются действия пользователя, представленные наборами событий MotionEvents. С помощью класса GestureDetector приложение может реагировать на более сложные пользовательские жесты, такие как *щелчки*, *двойные нажатия*, *длительные нажатия* и *прокрутки*. Приложения могут «отвечать» на подобные события с помощью реализации методов интерфейсов GestureDetector.OnGestureListener и GestureDetector.OnDoubleTapListener. Класс GestureDetector.SimpleOnGestureListener представляет собой класс адаптера, который реализует все методы для двух вышеперечисленных интерфейсов. В результате появляется возможность расширения класса и переопределения требуемого метода(ов), относящегося к этим интерфейсам. В разделе 7.5.2 будет инициализирован класс GestureDetector вместе с методом SimpleOnGestureListener, используемым для обработки событий двойного касания экрана, вызывающих выстрел из пушки.

Добавление звука с помощью SoundPool и AudioManager

Звуковыми эффектами приложения можно управлять с помощью класса SoundPool (пакет android.media). Доступны загрузка, воспроизведение и выгрузка звуков. Для воспроизведения звуков используется один из нескольких аудиопотоков Android, включающих потоки для оповещений, тонов DTMF, музыки, уведомлений, телефонных звонков, системных звуков и рингтонов. В документации Android рекомендуется использовать для воспроизведения звука в играх музыкальный аудиопоток. С помощью метода setVolumeControlStream класса Activity определяется возможность управления громкостью звука в игре кнопками регулировки громкости устройства, которые также используются для изменения громкости воспроизведения музыки. Метод получает константу из класса AudioManager (пакет android.media).

Покадровая анимация с помощью потоков, SurfaceView и SurfaceHolder

Это приложение реализует *выполнение анимации вручную* путем обновления элементов игры, находящихся в отдельном потоке выполнения. Для этого используется подкласс класса Thread с методом run, который с помощью пользовательского класса CannonView обновляет позиции всех элементов игры, а затем обновляет сами элементы. Как правило, любые обновления пользовательского интерфейса пользователя должны выполняться с помощью потока выполнения GUI. Но в Android важно минимизировать

объем работы, выполняемой в потоке GUI, чтобы убедиться в том, что GUI остается «отзывчивым» и не отображает диалоговые окна ANR (Application Not Responding, Приложение не отвечает).

Зачастую в играх задействована сложная логика, которая должна работать в отдельных потоках выполнения, иногда эти потоки требуется отображать на экране. Для подобных случаев Android поддерживает класс SurfaceView — подкласс класса View, в котором может быть «нарисован» произвольный поток. Классом SurfaceView можно манипулировать посредством объекта класса SurfaceHolder, позволяющего получить объект Canvas, на котором рисуется графика. Класс SurfaceHolder также поддерживает методы, обеспечивающие потоку исключительный доступ к объекту Canvas для рисования, поскольку рисовать в SurfaceView может лишь один поток одновременно. Каждый подкласс SurfaceView должен реализовывать интерфейс SurfaceHolder.Callback, включающий методы, которые вызываются при создании, изменять (размеры или ориентацию) либо уничтожать компонент SurfaceView.

Простое обнаружение конфликтов

С помощью класса CannonView выполняется простое обнаружение столкновений пушечного ядра с каким-либо краем компонента CannonView, с блоком или с секцией мишени. Эти методики представлены в разделе 7.5.3.

ПРИМЕЧАНИЕ

> Многие фреймворки, предназначенные для разработки игр, предлагают более сложные механизмы обнаружения столкновений.

Рисование графики с помощью Paint и Canvas

Методы класса Canvas (пакет android.graphics) применяются для рисования текста, линий и окружностей. Метод Canvas рисует в области объекта Bitmap класса View. Каждый метод рисования класса Canvas использует объект класса Paint (пакет android.graphics) для определения характеристик рисования, включая цвет, толщину линии, размер шрифта и другие параметры. Эти характеристики реализованы с помощью метода drawGameElements, описанного в разделе 7.5.3. Чтобы получить дополнительные сведения о характеристиках, используемых в процессе рисования и указываемых с помощью объекта Paint, посетите веб-сайт developer.android.com/reference/android/graphics/Paint.html.

7.4. Создание графического интерфейса пользователя приложения и файлов ресурсов

В этом разделе будут созданы файлы ресурсов приложения и файл разметки main.xml.

7.4.1. Создание проекта

Начните создание нового проекта Android под названием Cannon Game. В диалоговом окне New Android Project (Новый проект Android) введите следующие значения, а затем нажмите кнопку Finish (Готово):

- Build Target (Операционная система): Android 2.3.3.
- Application name (Имя приложения): Cannon Game.
- Package name (Название пакета): com.deitel.cannongame.
- Create Activity (Создать деятельность): CannonGame.
- Min SDK Version (Минимальная версия SDK): 8.

7.4.2. Файл AndroidManifest.xml

В листинге 7.1 приведен XML-код из файла приложения AndroidManifest.xml. Как и в разделе 6.6, атрибуту элемента деятельности android:screenOrientation было присвоено значение "portrait" (строка 9), в результате чего приложение всегда отображается в портретном режиме.

Листинг 7.1. Файл AndroidManifest.xml

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <manifest xmlns:android=http://schemas.android.com/apk/res/android
 3    package="com.deitel.cannongame" android:versionCode="1"
 4    android:versionName="1.0">
 5    <application android:icon="@drawable/icon"
 6       android:label="@string/app_name" android:debuggable="true">
 7       <activity android:name=".CannonGame"
 8          android:label="@string/app_name"
 9          android:screenOrientation="portrait">
10          <intent-filter>
11             <action android:name="android.intent.action.MAIN" />
12             <category android:name="android.intent.category.LAUNCHER" />
13          </intent-filter>
14       </activity>
15    </application>
16    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="10"/>
17 </manifest>
```

7.4.3. Файл strings.xml

В файле strings.xml, относящемся к приложению, определяются форматные строки (листинг 7.2, строки 4–5 и 9–10). Как упоминалось в разделе 7.3, в форматных строках, включающих несколько спецификаторов формата, эти спецификаторы нумеруются (например, для задач локализации приложения). Запись 1$ в спецификаторе %1$.1f (строка 5) указывает на то, что первый аргумент после форматной строки должен заменять спецификатор формата %1$d. Аналогично, запись %2$.1f указывает на то, что второй аргумент после форматной строки должен заменять спецификатор формата %2$.1f. Буква d в первом спецификаторе формата указывает на то, что выполняется форматирование в виде десятичного целого числа, а буква f во втором спецификаторе формата определяет форматирование в виде значения с плавающей точкой. В локализованных версиях файла strings.xml спецификаторы формата %1$d и %2$.1f могут переупорядочиваться в случае необходимости. То есть первый аргумент после форматной строки в вызове метода format класса Resourcesmethod getString или String может

заменять %1$d, а второй аргумент может заменять %2$.1f (независимо от того, где они появляются в форматной строке).

Листинг 7.2. Строки, определенные в файле strings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
   <string name="app_name">Cannon Game</string>
   <string name="results_format">
      Shots fired: %1$d\nTotal time: %2$.1f</string>
   <string name="reset_game">Reset Game</string>
   <string name="win">You win!</string>
   <string name="lose">You lose!</string>
   <string name="time_remaining_format">
      Time remaining: %.1f seconds</string>
</resources>
```

7.4.4. Файл main.xml

При создании приложения из этой главы удален файл main.xml, вместо которого подставлен файл, включающий FrameLayout. Единственный компонент в макете приложения — экземпляр пользовательского подкласса View под названием CannonView добавлен в проект в разделе 7.5.3. В листинге 7.3 приведен завершенный файл main.xml, в который вручную добавлен XML-элемент, показанный в строках 2–7. Этот элемент определяет, что компонент CannonView занимает всю ширину и высоту родительского макета, а также имеет белый фон. В разделе 7.3 вам уже приходилось полностью определять имя пользовательского класса View в разметке XML, где в строке 2 содержится ссылка на класс CannonView как на com.deitel.cannongame.CannonView.

Листинг 7.3. XML-разметка (main.xml) приложения Cannon Game

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <com.deitel.cannongame.CannonView
3     xmlns:android=http://schemas.android.com/apk/res/android
4     android:id="@+id/cannonView"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:background="@android:color/white"/>
```

7.4.5. Добавление звуков в приложение

Как уже упоминалось ранее, звуковые файлы хранятся в папке приложения res/raw. В рассматриваемом приложении используются три звуковых файла: blocker_hit.wav, target_hit.wav и cannon_fire.wav. Эти файлы находятся в папке sounds примеров к книге. Чтобы добавить эти файлы в проект, выполните следующие действия:

1. Щелкните правой кнопкой мыши на папке res приложения и выполните команды New ▸ Folder (Создать ▸ Папка).
2. Укажите имя папки raw и щелкните на кнопке Finish (Готово), чтобы создать папку.
3. Перетащите файлы звуков в папку res/raw.

7.5. Создание приложения

Это приложение включает три класса — Line (листинг 7.4), CannonGame (подкласс Activity; листинги 7.5–7.8) и CannonView (листинги 7.9–7.21).

7.5.1. Определение концов линии с помощью класса Line

Класс Line (см. листинг 7.4) просто группирует две точки Points, которые представляют начальную и конечную точки линии (Point). Объекты из этого класса будут использованы для определения блока и мишени. Чтобы добавить класс Line в проект, выполните следующие действия:

1. Раскройте узел проекта src в окне обозревателя Package Explorer.
2. Щелкните правой кнопкой мыши на пакете (com.deitel.cannongame) и в контекстном меню выберите команды New ▸ Class (Создать ▸ Класс) для отображения диалогового окна New Java Class (Новый класс Java).
3. В поле Name (Имя), которое отображается в диалоговом окне, введите Line, а затем щелкните на кнопке Finish (Готово).
4. Введите код, представленный в листинге 7.4, в файл Line.java.

Листинг 7.4. Класс Line представляет линию двумя концами

```
// Line.java
// Класс Line, представляющий
// линию двумя концами.
package com.deitel.cannongame;

import android.graphics.Point;

public class Line
{
   public Point start; // начальная точка
   public Point end;   // конечная точка

   // заданный по умолчанию конструктор, который присваивает
   // объектам Point (точки) начальные
   // значения координат (0, 0)
   public Line()
   {
      start = new Point(0, 0); // начальная точка
      end = new Point(0, 0);   // конечная точка
   }  // конец метода Line
}  // конец класса Line
```

7.5.2. Подкласс CannonGame класса Activity

Класс CannonGame (см. листинги 7.5–7.8) является главным классом Activity приложения Cannon Game.

Операторы package, import и экземпляры переменных

В разделе 7.3 были рассмотрены новые ключевые классы и интерфейсы, которые использует класс CannonGame. Эти классы и интерфейсы выделены в листинге 7.5. В строке 15 определяется переменная cannonView, с помощью которой осуществляется взаимодействие между классами CannonGame и CannonView.

Листинг 7.5. Операторы package, import и экземпляры переменных класса CannonGame

```
 1 // CannonGame.java
 2 // Главный класс Activity для приложения Cannon Game.
 3 package com.deitel.cannongame;
 4
 5 import android.app.Activity;
 6 import android.os.Bundle;
 7 import android.media.AudioManager;
 8 import android.view.GestureDetector;
 9 import android.view.MotionEvent;
10 import android.view.GestureDetector.SimpleOnGestureListener
11
12 public class CannonGame extends Activity
13 {
14    private GestureDetector gestureDetector; // прослушивание
                                               // двойных тапов
15    private CannonView cannonView; // пользовательское представление
                                     //экрана игры
16
```

Переопределение методов onCreate, onPause и onDestroy класса Activity

В листинге 7.6 представлены перегруженные методы onCreate (строки 18–32), onPause (строки 35–40) и onDestroy (строки 43–48) класса Activity. Метод onCreate создает структуру main.xml деятельности, затем получает ссылку на объект CannonView (строка 25). В строке 28 создается переменная GestureDetector, которая идентифицирует двойные тапы для текущей деятельности с помощью метода gestureListener, определенного в листинге 7.8. Оператор, находящийся в строке 31, обеспечивает регулирование громкости звукового сопровождения игры с помощью кнопок настройки громкости устройства Android.

Листинг 7.6. Переопределение методов onCreate, onPause и onDestroy класса Activity

```
17    // вызывается при первом запуске приложения
18    @Override
19    public void onCreate(Bundle savedInstanceState)
20    {
21       super.onCreate(savedInstanceState); // вызов метода onCreate класса super
22       setContentView(R.layout.main);   // создание структуры
23
24       // получение CannonView
25       cannonView = (CannonView) findViewById(R.id.cannonView);
```

продолжение ☞

Листинг 7.6 *(продолжение)*

```
26
27        // Инициализация GestureDetector
28        gestureDetector = new GestureDetector(this, gestureListener);
29
30        // Настройка громкости игры с помощью кнопок
          // регулирования громкости устройства
31        setVolumeControlStream(AudioManager.STREAM_MUSIC);
32    }  // конец метода onCreate
33
34    // пауза в случае, если приложение выполняется в фоновом режиме
35    @Override
36    public void onPause()
37    {
38       super.onPause();       // вызов метода super
39       cannonView.stopGame(); // завершение игры
40   }   // конец метода onPause
41
42   // освобождение ресурсов
43   @Override
44   protected void onDestroy()
45   {
46      super.onDestroy();
47      cannonView.releaseResources();
48   }  // конец метода onDestroy
49
```

Метод onPause (строки 35–40) гарантирует, что деятельность CannonGame не будет выполняться, если она находится в фоновом режиме. Если игра продолжает выполняться, пользователь не может взаимодействовать с ней в силу того, что другая деятельность находится в фокусе. Но в этом случае все равно потребляется энергия аккумуляторной батареи — самый ценный ресурс для мобильных устройств. При вызове метода onPause в строке 39 вызывается метод stopGame класса cannonView (см. листинг 7.19). Этот метод выполняет завершение потока игры — в этом примере невозможно сохранить состояние игры.

Как только деятельность будет завершена, метод onDestroy (строки 43–46) вызывает метод releaseResources класса cannonView (см. листинг 7.19), который освобождает ресурсы звука приложения.

Переопределение метода onTouchEvent класса Activity

В этом примере будет выполнено переопределение метода onTouchEvent (см. листинг 7.7) для определения момента касания пальцем экрана или перемещения пальца вдоль экрана. Параметр MotionEvent содержит информацию о происшедшем событии. В строке 55 используется метод getAction класса MotionEvent для определения типа происходящего события. В строках 58–59 идентифицируется факт касания экрана пользователем (MotionEvent.ACTION_DOWN) либо перемещения пальца вдоль экрана (MotionEvent.ACTION_MOVE). В том или ином случае в строке 61 вызывается метод alignCannon класса cannonView (см. листинг 7.16), с помощью которого выполняется нацеливание пушки

в точку касания экрана пальцем. В строке 65 объект MotionEvent передается методу onTouchEvent класса gestureDetector для определения, имел ли место двойной тап.

Листинг 7.7. Переопределение метода onTouchEvent класса Activity

```
50   // вызывается, если пользователь касается экрана
     // при выполняемой деятельности
51   @Override
52   public Boolean OnTouchEvent(MotionEvent event)
53   {
54      // тип int, представляющий действие, вызываемое этим событием
55      int action = event.getAction();
56
57      // пользователь коснулся экрана или провел пальцем по нему
58      if (action == MotionEvent.ACTION_DOWN ||
59      action == MotionEvent.ACTION_MOVE)
60      {
61         cannonView.alignCannon(event); // нацеливание пушки
62      }  // end if
63
64      // вызов метода onTouchEvent класса GestureDetector
65      return gestureDetector.onTouchEvent(event);
66   } // конец метода onTouchEvent
```

Анонимный внутренний класс, расширяющий класс SimpleOnGestureListener

Код из листинга 7.8 создает класс SimpleOnGestureListener под именем gestureListener, который регистрируется в строке 28 с помощью GestureDetector. Класс адаптера SimpleOnGestureListener реализует все методы интерфейсов OnGestureListener и OnDoubleTapListener. Эти методы возвращают значение false, свидетельствующее о том, что события не обрабатываются. Мы переопределяем лишь метод onDoubleTap (строки 71–76), вызывающийся после двойного тапа экрана пользователем. В строке 74 вызывается метод fireCannonBall класса CannonView (см. листинг 7.15), который выстреливает пушечное ядро. Метод fireCannonBall получает координаты двойного тапа пальцем экрана с помощью аргумента MotionEvent. Эти сведения используются для нацеливания пушки под правильным углом. В строке 75 возвращается значение true, свидетельствующее, что событие было обработано.

Листинг 7.8. Анонимный внутренний класс, расширяющий класс SimpleOnGestureListener

```
67      // прослушивание событий, связанных с касаниями,
        // отправленных GestureDetector
68      SimpleOnGestureListener gestureListener =
           new SimpleOnGestureListener()
69      {
70         // вызывается после двойного тапа пользователем экрана
71         @Override
72         public boolean onDoubleTap(MotionEvent e)
73         {
```

продолжение ⇯

Листинг 7.8 *(продолжение)*

```
            cannonView.fireCannonball(e); // стрельба ядром
            return true; // событие было обработано
         }  // конец метода onDoubleTap
      }; // конец gestureListener
   }  // конец класса CannonGame
```

7.5.3. Подкласс CannonView класса View

Класс CannonView (см. листинги 7.9–7.21) — это пользовательский подкласс класса View, который реализует логику игры Cannon Game и рисует объекты игры на экране. Чтобы добавить класс в проект, выполните следующие действия:

1. Раскройте узел src проекта в окне Package Explorer.
2. Щелкните правой кнопкой мыши на пакете (com.deitel.cannongame) и выполните команды New ▸ Class (Создать ▸ Класс) для отображения диалогового окна New Java Class (Новый класс Java).
3. В поле Name (Имя) диалогового окна введите название CannonView, в поле Superclass (Суперкласс) введите android.view.View, а затем щелкните на кнопке Finish (Готово).
4. Введите код, приведенный в листингах 7.9–7.19, в файл CannonView.java.

Операторы package и import

В листинге 7.9 приведены операторы package и import для класса CannonView. В разделе 7.3 рассмотрены новые ключевые классы и интерфейсы, используемые классом CannonView.

Листинг 7.9. Операторы package и import класса CannonView

```
// CannonView.java
// Отображение Cannon Game
package com.deitel.cannongame;

import java.util.HashMap;
import java.util.Map;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import android.media.AudioManager;
import android.media.SoundPool;
import android.util.AttributeSet;
import android.view.MotionEvent;
```

```
import android.view.SurfaceHolder;
import android.view.SurfaceView;

```

Переменные и константы экземпляра класса CannonView

В листинге 7.10 перечислены константы и переменные экземпляра класса `CannonView`. Большинство из них не требуют дополнительных объяснений, но тем не менее будут рассмотрены по мере их применения в коде.

Листинг 7.10. Поля класса CannonView

```
public class CannonView extends SurfaceView
   implements SurfaceHolder.Callback

{
   private CannonThread cannonThread; // контроль цикла игры
   private Activity activity;    // отображение диалогового окна
                                 // Game Over в потоке GUI
   private boolean dialogIsDisplayed = false;

   // константы, используемые в игре
   public static final int TARGET_PIECES = 7;  // секции мишени
   public static final int MISS_PENALTY = 2;   // секунды, вычитаемые
                                               // в случае промаха
   public static final int HIT_REWARD = 3;     // секунды, добавляемые
                                               // при попадании

   // переменные, используемые для цикла игры и отслеживания статистики
   private boolean gameOver; // игра завершена?
   private double timeLeft; // оставшееся время в секундах
   private int shotsFired; // количество выстрелов пользователя
   private double totalTimeElapsed; // количество прошедших секунд

   // переменные, используемые при определении блока и цели
   private Line blocker; // начальная и конечная точки блока
   private int blockerDistance;  // расстояние от блока слева
   private int blockerBeginning; // расстояние от блока сверху
   private int blockerEnd; // расстояние от нижнего края блока сверху
   private int initialBlockerVelocity; // множитель начальной
                                       // скорости блока
   private float blockerVelocity;  // множитель скорости блока
                                   // во время игры

   private Line target; // начальная и конечная точки мишени
   private int targetDistance;  // дистанция до мишени слева
   private int targetBeginning; // дистанция до мишени сверху
   private double pieceLength;  // длина секции мишени
   private int targetEnd; // дистанция сверху до нижнего края мишени
   private int initialTargetVelocity; // множитель начальной скорости
                                      // мишени
```

продолжение ⇨

Листинг 7.10 *(продолжение)*

```
    private float targetVelocity;  // множитель скорости мишени
                                   // во время игры
    private int lineWidth; // ширина мишени и блока
    private boolean[] hitStates; // попадание во все секции мишени?
    private int targetPiecesHit; // количество пораженных секций
                                 // мишени (из 7)
    // переменные для пушки и пушечного ядра
    private Point cannonball; // изображение ядра в левом верхнем углу
    private int cannonballVelocityX; // скорость ядра по горизонтали
    private int cannonballVelocityY; // скорость ядра по вертикали
    private boolean cannonballOnScreen; // ядро на экране
    private int cannonballRadius; // радиус ядра
    private int cannonballSpeed;  // скорость ядра
    private int cannonBaseRadius; // радиус основания пушки
    private int cannonLength; // длина ствола пушки
    private Point barrelEnd;  // конец ствола пушки
    private int screenWidth;  // ширина экрана
    private int screenHeight; // высота экрана
    // константы и переменные, используемые для управления звуком
    private static final int TARGET_SOUND_ID = 0;
    private static final int CANNON_SOUND_ID = 1;
    private static final int BLOCKER_SOUND_ID = 2;
    private SoundPool soundPool; // звуковые эффекты
    private Map<Integer, Integer> soundMap; // отображение ID
                                            // на SoundPool
    // переменные Paint, используемые для рисования на экране
    private Paint textPaint; // Paint, используемая для рисования текста
    private Paint cannonballPaint; // Paint, используемая
                                   // для рисования ядра
    private Paint cannonPaint;     // Paint, используемая
                                   // для рисования пушки
    private Paint blockerPaint;    // Paint, используемая
                                   // для рисования блока
    private Paint targetPaint;     // Paint, используемая
                                   // для рисования мишени
    private Paint backgroundPaint; // Paint, используемая
                                   // для чистки области рисования
```

Конструктор класса CannonView

Код листинга 7.11 определяет конструктор класса CannonView. После «раздувания» класса View вызывается его конструктор, а в качестве аргументов передаются Context и AttributeSet. В этом случае Context представляет собой класс Activity (CannonGame), к которому присоединен CannonView, а AttributeSet (пакет android.util) содержит значения для любых атрибутов, установленных в разметке XML-документа. Эти аргументы должны

быть переданы конструктору суперкласса (строка 92) для обеспечения правильного конфигурирования пользовательского объекта View с помощью значений для любых стандартных атрибутов View, определенных в XML.

В строке 93 хранится ссылка на родительский класс Activity, который можно использовать в конце игры для отображения окна AlertDialog из потока Activity GUI. В строке 96 выполняется регистрация класса (то есть CannonView) в качестве объекта, который реализует SurfaceHolder.Callback для получения вызовов метода, которые свидетельствуют о том, что класс SurfaceView создан, обновлен и уничтожен. Метод getHolder класса SurfaceView возвращает соответствующий объект SurfaceHolder, используемый для управления SurfaceView, и метод addCallback класса SurfaceHolder, который хранит объект, реализующий метод SurfaceHolder.Callback.

Листинг 7.11. Конструктор класса CannonView

```
 89 // общедоступный конструктор
 90 public CannonView(Context context, AttributeSet attrs)
 91 {
 92    super(context, attrs); // вызов конструктора super
 93    activity = (Activity) context;
 94
 95    // регистрация слушателя SurfaceHolder.Callback
 96    getHolder().addCallback(this);
 97
 98    // инициализация линий и точек, представляющих элементы игры
 99    blocker = new Line();     // создание блока в виде линии
100    target = new Line();      // создание мишени в виде линии
101    cannonball = new Point(); // создание ядра в виде точки
102
103    // инициализация hitStates в виде булевского массива
104    hitStates = new boolean[TARGET_PIECES];
105
106    // инициализация SoundPool, используемого для воспроизведения
       // трех звуковых эффектов
107    soundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);
108
109    // создание карты звуков и предварительная загрузка звуков
110    soundMap = new HashMap<Integer, Integer>(); // создание
                                                // нового HashMap
111    soundMap.put(TARGET_SOUND_ID,
112       soundPool.load(context, R.raw.target_hit, 1));
113    soundMap.put(CANNON_SOUND_ID,
114       soundPool.load(context, R.raw.cannon_fire, 1));
115    soundMap.put(BLOCKER_SOUND_ID,
116       soundPool.load(context, R.raw.blocker_hit, 1));
117
118    // конструирует Paints для рисования текста, ядра, пушки
119    // блока и мишени; все они конфигурируются с помощью
       // метода onSizeChanged
120    textPaint = new Paint();       // Paint для рисования текста
121    cannonPaint = new Paint();     // Paint для рисования пушки
```

продолжение ⇨

Листинг 7.11 *(продолжение)*

```
122      cannonballPaint = new Paint(); // Paint для рисования ядра
123      blockerPaint = new Paint();    // Paint для рисования блока
124      targetPaint = new Paint();     // Paint для рисования мишени
125      backgroundPaint = new Paint(); // Paint для рисования фона
126   }  // конец конструктора CannonView
127
```

В строках 99–101 создается блок и мишень с помощью объектов Lines и пушечное ядро с помощью объекта Point. Затем создается булев массив hitStates, предназначенный для отслеживания попаданий в секции мишени (которые при этом исчезают с экрана).

В строках 107–116 конфигурируются звуки, используемые в приложении. Сначала создается объект SoundPool, применяемый для загрузки и воспроизведения звуковых эффектов, используемых в приложении. Первый аргумент представляет максимальное количество звуковых потоков, которые могут воспроизводиться одновременно. В игре воспроизводится лишь один звук, поэтому значение этого аргумента равно 1. Второй аргумент определяет аудиопоток, который будет использоваться для воспроизведения звуков. Используются семь аудиопотоков, идентифицируемых константами класса AudioManager. Обратите внимание, что в документации для класса SoundPool рекомендуется использовать поток воспроизведения музыки (AudioManager.STREAM_MUSIC) для воспроизведения звука в играх. Последний аргумент представляет качество звука, хотя в документации указывается, что в данное время это значение не используется (в качестве используемого по умолчанию значения указано 0).

В строке 110 создается объект HashMap (переменная soundMap). В строках 111–116 задаются значения с помощью констант, используемых в качестве ключей (строки 75–77). Соответствующие значения представляют собой возвращаемые значения метода load класса SoundPool, возвращающие идентификатор ID, который может использоваться для воспроизведения (или выгрузки) звука. Метод load класса SoundPool получает три аргумента — контент (Context) приложения, идентификатор (ID), представляет загружаемый звуковой файл, а также приоритет (priority) звука. В соответствии с документацией, описывающей этот метод, последний аргумент в настоящее время не используется и имеет значение 1.

В строках 120–125 создаются объекты Paint, используемые для рисования объектов игры. Эти объекты конфигурируются с помощью метода onSizeChanged, поскольку некоторые из настроек объекта Paint зависят от масштаба элементов игры, основанном на размере экрана устройства.

Overriding View Method onSizeChanged

В коде из листинга 7.12 выполняется переопределение метода onSizeChanged класса View, который вызывается в случае изменения размеров класса View, например при первом добавлении класса View в иерархию классов View в процессе «раздувания» разметки. Это приложение всегда отображается в портретном режиме, поэтому метод onSizeChanged вызывается только в том случае, если метод onCreate деятельности «раздувает» GUI. Этот метод принимает новые значения ширины и высоты класса View. Прежним значениям ширины и высоты (при первом вызове метода) присваиваются значения 0. При

выполнении вычислений, реализующих масштабирование изображения на экране, используются размеры экрана устройства (ширина и высота, выраженные в пикселях). С помощью метода «проб и ошибок» получаем значения множителей масштабирования. После выполнения вычислений в строке 173 вызывается метод newGame (см. листинг 7.13).

Листинг 7.12. Переопределенный метод onSizeChanged

```
128  // вызывается при изменении размеров представления, а также
129  // при первом добавлении в иерархию представлений
130  @Override
131  protected void onSizeChanged(int w, int h, int oldw, int oldh)
132  {
133     super.onSizeChanged(w, h, oldw, oldh);
134
135     screenWidth = w;  // хранение значения ширины
136     screenHeight = h; // хранение значения высоты
137     cannonBaseRadius = h / 18; // радиус основания пушки, равный
                                 // 1/18 от высоты экрана
138     cannonLength = w / 8; // длина пушки равна 1/8 от ширины экрана
139
140     cannonballRadius = w / 36; // радиус ядра равен 1/36 ширины экрана
141     cannonballSpeed = w * 3 / 2; // множитель скорости ядра
142
143     lineWidth = w / 24; // ширина мишени и блока равны 1/24
                          // от ширины экрана
144
145     // конфигурирование переменных экземпляра, связанных с блоком
146     blockerDistance = w * 5 / 8; // зазор слева от блока равен
                                   // 5/8 ширины экрана
147     blockerBeginning = h / 8; // зазор сверху от блока равен
                                // 1/8 от высоты экрана
148     blockerEnd = h * 3 / 8; // зазор сверху от блока равен 3/8
                              // от высоты экрана
149     initialBlockerVelocity = h / 2; // начальный множитель
                                      // скорости блока
150     blocker.start = new Point(blockerDistance, blockerBeginning);
151     blocker.end = new Point(blockerDistance, blockerEnd);
152
153     // конфигурирование переменных экземпляра, связанных с мишенью
154     targetDistance = w * 7 / 8; // зазор слева от мишень равен
                                  // 7/8 от ширины экрана
155     targetBeginning = h / 8; // зазор сверху от мишени равен
                               // 1/8 от высоты экрана
156     targetEnd = h * 7 / 8; // зазор сверху равен 7/8 от высоты экрана
157     pieceLength = (targetEnd – targetBeginning) / TARGET_PIECES;
158     initialTargetVelocity = -h / 4; // множитель начальной
                                      // скорости мишени
159     target.start = new Point(targetDistance, targetBeginning);
160     target.end = new Point(targetDistance, targetEnd);
161
```

продолжение ☞

Листинг 7.12 *(продолжение)*

```
   // конечная точка ствола пушки (изначально направлен по горизонтали)
   barrelEnd = new Point(cannonLength, h / 2);

   // конфигурирование объектов Paint для рисования элементов игры
   textPaint.setTextSize(w / 20); // размер текста равен 1/20
                                 // от ширины экрана
   textPaint.setAntiAlias(true);  // сглаживание текста
   cannonPaint.setStrokeWidth(lineWidth * 1.5f); // настройка
                                                // толщины линии
   blockerPaint.setStrokeWidth(lineWidth); // настройка толщины
                                          // линии
   targetPaint.setStrokeWidth(lineWidth);  // настройка толщины линии
   backgroundPaint.setColor(Color.WHITE);  // настройка фонового цвета

   newGame(); // первоначальная настройка и запуск новой игры
} // end method onSizeChanged

```

Метод newGame класса CannonView

Метод newGame (см. листинг 7.13) сбрасывает начальные значения переменных, используемых для управления игрой. Если значение переменной gameOver равно true, что бывает только после окончания первой игры, в строке 197 переустанавливается значение переменной gameOver, а в строках 198–199 создается и запускается новый поток CannonThread для начала новой игры.

Листинг 7.13. Метод newGame класса CannonView

```
176 // переустановка всех элементов экрана и запуск новой игры
177 public void newGame()
178 {
179    // присваивание каждому элементу массива hitStates значения
       //  false -- восстановление секций мишени
180    for (int i = 0; i < TARGET_PIECES; ++i)
181       hitStates[i] = false;
182
183    targetPiecesHit = 0; // нет попаданий в секции мишени
184    blockerVelocity = initialBlockerVelocity; // настройка
                                                // начальной скорости
185    targetVelocity = initialTargetVelocity;   // настройка
                                                // начальной скорости
186    timeLeft = 10; // начало обратного отсчета 10 секунд
187    cannonballOnScreen = false; // ядро не отображается на экране
188    shotsFired = 0; // установка начального числа выстрелов
189    totalElapsedTime = 0.0; // обнуление времени игры
190    blocker.start.set(blockerDistance, blockerBeginning);
191    blocker.end.set(blockerDistance, blockerEnd);
192    target.start.set(targetDistance, targetBeginning);
193    target.end.set(targetDistance, targetEnd);
194
```

```
      if (gameOver)
      {
         gameOver = false; // игра не завершена
         cannonThread = new CannonThread(getHolder());
         cannonThread.start();
      } // конец блока if
   } // конец метода newGame

```

Метод updatePositions класса CannonView

Метод updatePositions (см. листинг 7.14) вызывается методом run класса CannonThread (см. листинг 7.21), с помощью которого выполняется обновление положения элементов на экране и простое обнаружение столкновений. Новые положения элементов игры на экране вычисляются на основе времени (в миллисекундах), прошедшего между предыдущим и текущим кадром анимации. С помощью этого метода игра обновляет величину расстояния, на которое перемещается каждый элемент игры (при этом используется скорость обновления изображения на экране устройства). Эти вопросы будут разобраны более подробно при рассмотрении циклов игры (см. листинг 7.21).

Листинг 7.14. Метод updatePositions класса CannonView

```
   // вызывается повторно с помощью CannonThread для обновления
   // элементов игры
   private void updatePositions(double elapsedTimeMS)
   {
      double interval = elapsedTimeMS / 1000.0; // преобразование
                                                // в секунды

      if (cannonballOnScreen) // был ли текущий выстрел
      {
         // обновление положения ядра
         cannonball.x += interval * cannonballVelocityX;
         cannonball.y += interval * cannonballVelocityY;

         // проверка столкновения с блоком
         if (cannonball.x + cannonballRadius > blockerDistance &&
            cannonball.x – cannonballRadius < blockerDistance &&
            cannonball.y + cannonballRadius > blocker.start.y &&
            cannonball.y – cannonballRadius < blocker.end.y)
            {
               cannonballVelocityX *= -1; // реверсирование
                                          // направления полета ядра
               timeLeft -= MISS_PENALTY;  // штраф для пользователя

               // воспроизведение звука блока
               soundPool.play(soundMap.get(BLOCKER_SOUND_ID),
                  1, 1, 1, 0, 1f)
           }   // конец блока if

           // проверка столкновений с левой и правой стенками
```

продолжение ☞

Листинг 7.14 *(продолжение)*

```
          else if (cannonball.x + cannonballRadius > screenWidth ||
             cannonball.x — cannonballRadius < 0)
             cannonballOnScreen = false; // remove
                        // изображения ядра с экрана

          // проверка столкновений с верхней и нижней стенками
          else if (cannonball.y + cannonballRadius > screenHeight ||
             cannonball.y — cannonballRadius < 0)
             cannonballOnScreen = false; // скрытие
                                         // изображения ядра

          // проверка столкновения ядра с мишенью
          else if (cannonball.x + cannonballRadius > targetDistance &&
             cannonball.x — cannonballRadius < targetDistance &&
             cannonball.y + cannonballRadius > target.start.y &&
             cannonball.y — cannonballRadius < target.end.y)
             {
                // определение номера секции мишени (0 сверху)
                int section =
                  (int) ((cannonball.y — target.start.y) / pieceLength);

          // проверка попадания в секцию мишени
          if ((section >= 0 && section < TARGET_PIECES) &&
             !hitStates[section])
          {
             hitStates[section] = true;  // попадание в секцию
             cannonballOnScreen = false; // удаление ядра
             timeLeft += HIT_REWARD;     // добавление
                                         // дополнительного времени

          // воспроизведение звука столкновения с целью
          soundPool.play(soundMap.get(TARGET_SOUND_ID), 1,
              1, 1, 0, 1f);

          // если были попадания во все секции мишени
          if (++targetPiecesHit == TARGET_PIECES)
          {
             cannonThread.setRunning(false);
             showGameOverDialog(R.string.win); // показать
                                  // диалоговое окно выигрыша
              gameOver = true; // игра завершена
          }   // конец блока if
      }   // конец блока if
  }   // конец блока else if
} // конец блока if

  // обновление позиции блока
  double blockerUpdate = interval * blockerVelocity;
  blocker.start.y += blockerUpdate;
  blocker.end.y += blockerUpdate;
```

```
274
275    // обновление позиции мишени
276    double targetUpdate = interval * targetVelocity;
277    target.start.y += targetUpdate;
278    target.end.y += targetUpdate;
279
280    // при попадании в верхнюю или нижнюю часть блока
       // реверсировать направление движения
281    if (blocker.start.y < 0 || blocker.end.y > screenHeight)
282       blockerVelocity *= -1;
283
284    // при попадании в верхнюю или нижнюю часть блока
       // реверсировать направление движения
285    if (target.start.y < 0 || target.end.y > screenHeight)
286       targetVelocity *= -1;
287
288    timeLeft -= interval; // вычитание из оставшегося времени
289
290    // если показания таймера равны нулю
291       if (timeLeft <= 0)
292       {
293          timeLeft = 0.0;
294          gameOver = true; // игра завершена
295          cannonThread.setRunning(false);
296          showGameOverDialog(R.string.lose); // диалоговое окно проигрыша
297       }  // end if
298    }  // конец метода updatePositions
299
```

В строке 206 преобразуется значение времени, прошедшего с момента последнего кадра анимации (от нескольких миллисекунд до секунд). Это значение используется для изменения позиций различных элементов игры.

В строке 208 проверяется, отображается ли пушечное ядро на экране. Если ядро отображается на экране, происходит обновление его позиции путем добавления расстояния, которое должно быть пройдено с момента последнего события таймера. Расстояние вычисляется путем умножения значения скорости на величину прошедшего времени (строки 211–212). В строках 215–218 проверяется, было ли столкновение ядра с блоком. *Обнаружение столкновений мы будем производить*, основываясь на рамке вокруг ядра. В случае столкновения ядра с блоком должны быть выполнены четыре условия:

- Результат суммирования координаты *x* пушечного ядра и радиуса ядра должен быть больше, чем расстояние от блока до левой границы экрана (`blockerDistance`), строка 215. Это означает, что ядро достигло блока, «пролетев» расстояние, отделяющее блок от левой границы экрана.
- Результат вычитания радиуса ядра из значения координаты *x* этого же ядра должен быть меньше, чем расстояние от блока до левого края экрана (строка 216). Это гарантирует, что ядро еще не миновало блок.
- Часть ядра должна располагаться ниже, чем верхняя часть блока (строка 217).
- Часть ядра должна располагаться выше, чем нижняя часть блока (строка 218).

Если все эти условия выполняются, направление движения ядра *реверсируется* (строка 220), пользователю *начисляется штраф* путем *вычитания* значения переменной MISS_PENALTY из значения переменной timeLeft, затем вызывается метод play класса soundPool для воспроизведения звука, сопровождающего попадание в блок. В качестве ключа soundMap используется BLOCKER_SOUND_ID для поиска идентификатора (ID) звука в SoundPool.

Изображение ядра исчезает с экрана после столкновения с любой из сторон экрана. В строках 228–230 проверяется, столкнулось ли ядро с левой либо с правой стенкой, и если это так, изображение ядра удаляется с экрана. В строках 233–235 удаляется изображение ядра в случае, если ядро сталкивается с верхним или нижним краем экрана.

Затем выполняется проверка столкновения ядра с мишенью (строки 238–241). При этом проверяются условия, подобные условиям столкновения ядра с блоком. Если ядро попало в мишень, идентифицируется *секция* мишени, в которую попало ядро. В строках 244–245 определяется секция мишени, в которую попало ядро, путем деления расстояния между ядром и нижней частью мишени на длину секции мишени. Значение этого выражения равно 0 для самой верхней секции мишени, и 6 — для самой нижней секции мишени. Чтобы проверить попадание ядра в секцию мишени, используется массив hitStates (строка 249). Если попадание имело место, соответствующему элементу массива hitStates присваивается значение true, а изображение ядра удаляется с экрана. Затем значение переменной HIT_REWARD добавляется в счетчик timeLeft, увеличивая тем самым значение оставшегося времени, и воспроизводится звук попадания в мишень (TARGET_SOUND_ID). Увеличивается значение переменной targetPiecesHit, определяется, равно ли оно значению переменной TARGET_PIECES (строка 260). Если это условие выполнено, игра завершается. При этом завершается выполнение потока CannonThread путем вызова метода setRunning с аргументом false, вызывается метод showGameOverDialog с идентификатором (ID) строкового ресурса, который определяет сообщение о выигрыше и присваивает переменной gameOver значение true.

Теперь, когда были проверены все возможные столкновения пушечного ядра, следует обновить значения позиций мишени и блока. В строках 271–273 изменяется значение позиции блока путем умножения значения переменной blockerVelocity на количество времени, которое прошло с момента последнего обновления и добавления значения к текущим координатам x и y. В строках 276–278 выполняются те же операции по отношению к мишени. Если блок сталкивается с верхней или нижней стенками, направление его движения реверсируется путем умножения значения скорости на -1 (строки 281–282). В строках 285–286 выполняются те же проверки и настройки для мишени, имеющей полную длину, включая секции мишени, в которые было попадание ядра.

Значение переменной timeLeft уменьшается на время, которое прошло с момента воспроизведения предыдущего кадра анимации. Если значение переменной timeLeft становится равным нулю, игра завершается. Переменной timeLeft присваивается значение 0.0 только в том случае, если она имеет отрицательное значение. Если этого не сделать, на экране может отобразиться отрицательное значение времени. Затем переменной gameOver присваивается значение true, прекращается выполнение потока CannonThread путем вызова метода setRunning с аргументом false, а также вызова метода showGameOverDialog с ID строкового ресурса, используемого для генерирования сообщения о проигрыше.

Метод fireCannonball класса CannonView

Если пользователь выполнит двойной тап экрана, обработчик событий для этого события (см. листинг 7.8) вызывает метод fireCannonball (см. листинг 7.15), с помощью которого выполняется выстрел из пушки. Если изображение ядра уже имеется на экране, метод тут же завершается. В противном случае (изображение ядра на экране отсутствует) происходит выстрел из пушки. В строке 306 вызывается метод alignCannon, с помощью которого нацеливается пушка в точку двойного тапа, а также считывается значение угла наклона пушки. В строках 309–310 «загружается» пушка (то есть положение пушечного ядра внутри пушки). Затем в строках 313 и 316 вычисляются горизонтальная и вертикальная компоненты скорости пушечного ядра. Затем переменной cannonballOnScreen присваивается значение true, в результате чего рисуется пушечное ядро с помощью метода drawGameElements (см. листинг 7.17) и увеличивается значение переменной shotsFired. И в заключение воспроизводится звук выстрела пушки (CANNON_SOUND_ID).

Листинг 7.15. Метод fireCannonball класса CannonView

```
300   // выстрел из пушки ядром
301   public void fireCannonball(MotionEvent event)
302   {
303      if (cannonballOnScreen) // если ядро отображается на экране
304         return; // ничего не делать
305
306      double angle = alignCannon(event); // получение угла наклона ствола
307
308      // помещение ядра вовнутрь ствола пушки
309      cannonball.x = cannonballRadius; // наклон пушки по x-координате
310      cannonball.y = screenHeight / 2; // центрирование ядра по вертикали
311
312      // получение x-компонента общей скорости
313      cannonballVelocityX = (int) (cannonballSpeed * Math.sin(angle));
314
315      // получение y-компонента общей скорости
316      cannonballVelocityY = (int) (-cannonballSpeed * Math.cos(angle));
317      cannonballOnScreen = true; // ядро отображено на экране
318      ++shotsFired; // увеличение значения переменной shotsFired
319
320      // воспроизведение звука выстрела из пушки
321      soundPool.play(soundMap.get(CANNON_SOUND_ID), 1, 1, 1, 0, 1f);
322  }   // end method fireCannonball
323
```

Метод alignCannon класса CannonView

Метод alignCannon (см. листинг 7.16) нацеливает пушку на точку двойного тапа экрана пользователем. В строке 328 выполняется получение координат x и y двойного тапа (с помощью аргумента MotionEvent). Вычисляется расстояние по вертикали от места касания до центра экрана. Если это расстояние не равно нулю, вычисляется угол наклона ствола пушки относительно горизонта (строка 338). Если касание экрана

произошло в его нижней части, настройка угла выполняется с помощью метода Math.PI (строка 342). С помощью переменных cannonLength и angle определяются координаты *x* и *y* для конечной точки ствола пушки. Затем рисуется линия в направлении от центра основания пушки, находящегося возле левого края экрана, до конечной точки ствола пушки.

Листинг 7.16. Метод alignCannon класса CannonView

```
324   // изменения угла наклона пушки в ответ на касания экрана
325   public double alignCannon(MotionEvent event)
326   {
327      // получение места касания в этом представлении
328      Point touchPoint = new Point((int) event.getX(),
            (int) event.getY());
329
330      // вычисление расстояния от места касания до центра
331      // экрана по оси y
332      double centerMinusY = (screenHeight / 2 – touchPoint.y);
333
334      double angle = 0; // инициализация угла значением 0
335
336      // вычисление угла наклона ствола относительно горизонта
337      if (centerMinusY != 0) // предотвращает деление на 0
338         angle = Math.atan((double) touchPoint.x / centerMinusY);
339
340      // в случае касания нижней половины экрана
341      if (touchPoint.y > screenHeight / 2)
342         angle += Math.PI; // изменение угла
343
344      // вычисление конечной точки ствола пушки
345      barrelEnd.x = (int) (cannonLength * Math.sin(angle));
346      barrelEnd.y =
347      (int) (-cannonLength * Math.cos(angle) + screenHeight / 2);
348
349      return angle; // возвращение вычисленного угла
350  }   // конец метода alignCannon
351
```

Рисование элементов игры

Метод drawGameElements (см. листинг 7.17) рисует пушку, пушечное ядро, блок и мишень с помощью класса SurfaceView. При этом используется компонент Canvas, который класс CannonThread получает из компонента SurfaceHolder класса SurfaceView.

Листинг 7.17. Метод drawGameElements класса CannonView

```
// рисование элементов игры с помощью выбранного Canvas
public void drawGameElements(Canvas canvas)
{
   // очистка фона
   canvas.drawRect(0, 0, canvas.getWidth(), canvas.getHeight(),
   backgroundPaint);

```

```
359      // отображение оставшегося времени
360      canvas.drawText(getResources().getString(
361         R.string.time_remaining_format, timeLeft), 30, 50, textPaint);
362
363      // если пушечное ядро на экране, нарисуйте его
364      if (cannonballOnScreen)
365         canvas.drawCircle(cannonball.x, cannonball.y, cannonballRadius,
366            cannonballPaint);
367
368      // рисование ствола пушки
369      canvas.drawLine(0, screenHeight / 2, barrelEnd.x, barrelEnd.y,
370         cannonPaint);
371
372      // рисование основания пушки
373      canvas.drawCircle(0, (int) screenHeight / 2,
374         (int) cannonBaseRadius, cannonPaint);
375
376      // рисование блока
377      canvas.drawLine(blocker.start.x, blocker.start.y, blocker.end.x,
378         blocker.end.y, blockerPaint);
379
380      Point currentPoint = new Point(); // начало текущей секции мишени
381
382      // присваивание curPoint начальной точки мишени
383      currentPoint.x = target.start.x;
384      currentPoint.y = target.start.y;
385
386      // рисование мишени
387      for (int i = 1; i <= TARGET_PIECES; ++i)
388      {
389         // если нет попадания в секцию мишени, нарисовать ее
390         if (!hitStates[i - 1])
391         {
392            // чередующееся раскрашивание частей мишени желтым и синим
393            if (i % 2 == 0)
394               targetPaint.setColor(Color.YELLOW);
395            else
396               targetPaint.setColor(Color.BLUE);
397
398            canvas.drawLine(currentPoint.x, currentPoint.y, target.end.x,
399               (int) (currentPoint.y + pieceLength), targetPaint);
400            }
401
402            // перемещение curPoint к началу следующей секции
403            currentPoint.y += pieceLength;
404   }  // конец метода for
405 } // конец метода drawGameElements
406
```

Сначала вызывается метод drawRect класса Canvas (строки 356–357), с помощью которого выполняется очистка объекта Canvas. В результате все элементы игры

отображаются в новых положениях. Этот метод получает в качестве аргументов координаты верхнего левого угла прямоугольника, ширину и высоту прямоугольника и объект Paint, определяющий характеристики рисования (вспомните, как с помощью backgroundPaint выбирался белый цвет рисования). Затем вызывается метод drawText класса Canvas (строки 360–361) для отображения оставшегося времени игры. В качестве аргументов передаются отображаемая строка, координаты x и y отображаемых объектов и компонент textPaint (сконфигурирован в строках 166–167), с помощью которого описывается отображаемый текст (размер шрифта, цвет и другие атрибуты текста).

Если пушечное ядро отображается на экране, в строках 365–366 используется метод drawCircle класса Canvas для рисования ядра в его текущей позиции. Первые два аргумента представляют координаты центра окружности. Третий аргумент представляет собой радиус окружности. Последний аргумент — это объект Paint, определяющий характеристики, используемые при рисовании окружности.

С помощью метода drawLine класса Canvas отображается ствол пушки (строки 369–370), блок (строки 377–378) и части мишени (строки 398–399). Этот метод получает пять параметров — первые четыре параметра представляют координаты x-y начала и конца линии, а последний — объект Paint, определяющий характеристики линии, например ее толщину.

В строках 373–374 используется метод drawCircle класса Canvas для рисования полукруглого основания пушки. При этом рисуется окружность, центр которой находится в левом углу экрана. И поскольку центр окружности находится в районе левого края экрана, половина окружности рисуется за пределами левой границы экрана компонента SurfaceView.

В строках 380–404 рисуются секции мишени. Осуществляется последовательный обход секций мишени, для каждой из которых используется корректный цвет (синий для секций с нечетными номерами и желтый для других секций). На экране отображаются секции мишени, в которые не попало ядро.

Метод showGameOverDialog класса CannonView

После завершения игры метод showGameOverDialog (см. листинг 7.18) выводит диалоговое окно AlertDialog, в котором отображается сообщение о том, выиграл или проиграл пользователь, показано количество произведенных выстрелов и общее время игры. В строках 419–430 вызывается метод setPositiveButton класса Builder, с помощью которого создается кнопка переустановки. Метод onClick слушателя кнопок сигнализирует о том, что диалоговое окно больше не отображается, и вызывает метод newGame для настройки и запуска новой игры. Диалоговое окно должно отображаться в потоке GUI, поэтому в строках 432–440 вызывается метод runOnUiThread класса Activity и передается объекту анонимного внутреннего класса, который реализует класс Runnable. Метод run класса Runnable проверяет возможность отображения диалогового окна и отображает его.

Листинг 7.18. Метод showGameOverDialog класса CannonView

```
407 // отображение диалогового окна AlertDialog после завершения игры
408 private void showGameOverDialog(int messageId)
409 {
410    // создание диалогового окна, отображающего данную строку
411    final AlertDialog.Builder dialogBuilder =
```

```
         new AlertDialog.Builder(getContext());
      dialogBuilder.setTitle(getResources().getString(messageId));
      dialogBuilder.setCancelable(false);

      // отображение количества выстрелов и общего времени игры
      dialogBuilder.setMessage(getResources().getString(
         R.string.results_format, shotsFired, totalElapsedTime));
      dialogBuilder.setPositiveButton(R.string.reset_game,
         new DialogInterface.OnClickListener()
         {
            // вызывается при нажатии кнопки "Reset Game"
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
               dialogIsDisplayed = false;
               newGame(); // настройка и запуск новой игры
            }  // конец метода onClick
        }   // конец анонимного внутреннего класса
     ); // конец вызова setPositiveButton

     activity.runOnUiThread(
        new Runnable() {
           public void run()
              {
                 dialogIsDisplayed = true;
                 dialogBuilder.show(); // отображение диалогового окна
              } // конец метода run
          }   // конец класса Runnable
       ); // конец вызова runOnUiThread
   }   // конец метода showGameOverDialog

```

Методы stopGame и releaseResources класса CannonView

Методы onPause и onDestroy (см. листинг 7.6) класса CannonGame (класс Activity) вызывают методы stopGame и releaseResources класса CannonView (см. листинг 7.19) соответственно. Метод stopGame (строки 444–448) вызывается из главного класса Activity и инициирует остановку игры после вызова метода onPause класса Activity. В целях упрощения в этом примере не хранятся сведения о состоянии игры. Метод releaseResources (строки 451–455) вызывает метод release класса SoundPool, с помощью которого выполняется освобождение ресурсов, связанных с SoundPool.

Листинг 7.19. Методы stopGame и releaseResources класса CannonView

```
443   // пауза в игре
444   public void stopGame()
445   {
446      if (cannonThread != null)
447      cannonThread.setRunning(false);
448   }  // конец метода stopGame
449
```

продолжение ⇨

Листинг 7.19 *(продолжение)*

```
// освобождает ресурсы; вызывается методом onDestroy
// класса CannonGame
public void releaseResources()
{
   soundPool.release(); // освобождает все ресурсы,
                        // используемые SoundPool
   soundPool = null;
}  // конец метода releaseResources

```

Реализация методов SurfaceHolder.Callback

В листинге 7.20 реализованы методы surfaceChanged, surfaceCreated и surfaceDestroyed интерфейса SurfaceHolder.Callback. Метод surfaceChanged является «пустышкой» в коде приложения, поскольку приложение всегда выводит изображение в портретном режиме. Этот метод вызывается в случае изменения размеров или ориентации компонента SurfaceView и обычно используется для повторного отображения графики после изменений. Метод surfaceCreated (строки 465–471) вызывается после создания SurfaceView (то есть в случае, если приложение сначала загружается или восстанавливается из приостановленного состояния). С помощью метода surfaceCreated создается и запускается поток CannonThread для начала выполнения игры. Метод surfaceDestroyed (строки 474–492) вызывается в случае разрушения SurfaceView (в случае завершения выполнения приложения). Метод CannonThread гарантирует корректное завершение приложения. Сначала в строке 479 вызывается метод setRunning класса CannonThread с аргументом false, это свидетельствует о том, что поток будет завершен. В строках 481–491 задается интервал ожидания до завершения потока. Это гарантирует, что после завершения выполнения метода surfaceDestroyed не будут предприниматься попытки рисования в области SurfaceView.

Листинг 7.20. Реализация методов SurfaceHolder.Callback

```
457  // вызывается в случае изменения размеров поверхности
458  @Override
459  public void surfaceChanged (SurfaceHolder holder, int format,
460     int width, int height)
461  {
462  }   // конец метода surfaceChanged
463
464  // вызывается при первом создании поверхности
465  @Override
466  public void SurfaceCreated (SurfaceHolder holder)
467  {
468     cannonThread = new CannonThread(holder);
469     cannonThread.setRunning(true);
470     cannonThread.start(); // начало потока цикла игры
471  }  // конец метода surfaceCreated
472
473  // вызывается при разрушении поверхности
474  @Override
475  public void surfaceDestroyed (SurfaceHolder holder)
```

```
   {
      // корректное завершение потока
      boolean retry = true;
      cannonThread.setRunning(false);

      while (retry)
      {
         try
         {
            cannonThread.join();
            retry = false;
         }  // конец блока try
         catch (InterruptedException e)
         {
      }  // конец блока catch
   }  // конец блока while
} // конец метода surfaceDestroyed

```

Поток CannonThread: использование потока для создания цикла игры

В листинге 7.21 определяется подкласс класса Thread при обновлении игры. Поток поддерживает ссылку на метод SurfaceHolder класса SurfaceView (строка 497), и с помощью булевого значения определяется, выполняется ли поток. Метод класса run (строки 514–543) управляет покадровыми анимациями (этот процесс известен как *цикл игры*). Каждое обновление элементов игры на экране выполняется на основании количества миллисекунд, прошедших с момента последнего обновления. В строке 518 считывается текущее системное время, выраженное в миллисекундах, прошедшее с момента начала выполнения потока. В строках 520–542 выполняется цикл до тех пор, пока значение переменной threadIsRunning не станет равным false.

Листинг 7.21. Класс Runnable, обновляющий игру через интервал времени, заданный переменной TIME_INTERVAL (в миллисекундах)

```
494    // Подкласс Thread, используемый для управления циклом игры
495    private class CannonThread extends Thread
496    {
497       private SurfaceHolder surfaceHolder; // управление canvas
498       private boolean threadIsRunning = true; // выполнение по умолчанию
499
500       // инициализация контейнера поверхности
501       public CannonThread(SurfaceHolder holder)
502       {
503          surfaceHolder = holder;
504          setName("CannonThread");
505       }  // конец конструктора
506
507       // изменение состояния выполнения
508       public void setRunning(boolean running)
509       {
510          threadIsRunning = running;
511       }  // конец метода setRunning
```

продолжение ➪

Листинг 7.21 *(продолжение)*

```
512
513      // управление циклом игры
514      @Override
515      public void run()
516      {
517         Canvas canvas = null; // используется для рисования
518         long previousFrameTime = System.currentTimeMillis();
519
520         while (threadIsRunning)
521         {
522            try
523            {
524               canvas = surfaceHolder.lockCanvas(null);
525
526               // блокирование surfaceHolder для рисования
527               synchronized(surfaceHolder)
528               {
529                  long currentTime = System.currentTimeMillis();
530                  double elapsedTimeMS = currentTime – previousFrameTime;
531                  totalElapsedTime += elapsedTimeMS / 1000.0;
532                  updatePositions(elapsedTimeMS); // обновление
                                                    //  состояния игры
533                  drawGameElements(canvas); // рисование
534                  previousFrameTime = currentTime; // обновление
                                                // предыдущего времени
535               }  // конец синхронизированного блока
536            }  // конец блока try
537            finally
538            {
539            if (canvas != null)
540               surfaceHolder.unlockCanvasAndPost(canvas);
541            }  // конец finally
542         }  // конец блока while
543      }  // конец метода run
544   }  // конец вложенного класса CannonThread
545 } // конец класса CannonView
```

Сначала получим объект Canvas, используемый для рисования, на SurfaceView путем вызова метода lockCanvas класса SurfaceHolder (строка 524). Для рисования на SurfaceView может использоваться единовременно лишь один поток, поэтому для выполнения синхронизации блока сначала нужно заблокировать SurfaceHolder. Затем снимаются показания текущего времени (в миллисекундах), вычисляется время выполнения программы и добавляется к счетчику суммарного времени. Это значение применяется для отображения количества оставшегося времени игры. В строке 532 вызывается метод updatePositions, аргументом которого является время выполнения приложения, заданное в миллисекундах. Этот метод перемещает все элементы игры, используя время выполнения приложения для облегчения масштабирования величины перемещения. В результате обеспечивается одна и та же скорость выполнения игры независимо от используемого устройства. Если интервал времени между соседними кадрами

увеличивается (при использовании более медленного устройства), элементы игры будут перемещаться на большее расстояние при отображении очередного кадра анимации. Если же интервал времени между кадрами меньше (для более быстрого устройства), элементы игры будут перемещаться на меньшее расстояние при отображении очередного кадра. И наконец, в строке 533 рисуются элементы игры с помощью метода Canvas класса SurfaceView, а в строке 534 хранится значение переменной currentTime в виде предыдущего previousFrameTime для подготовки вычисления времени выполнения приложения в следующем кадре анимации.

7.6. Резюме

В этой главе вы создали приложение Cannon Game, которое ставит перед игроком задачу разрушить состоящую из семи секций мишень до того, как истечет заданный интервал времени, равный 10 секундам. Пользователь наводить на цель пушку путем касания экрана. После двойного тапа экрана пушка выстреливает пушечным ядром.

Вы научились определять строковые ресурсы String для представления форматированных строк, которые используются для вызова метода getString класса Resource и метода format класса String, а также задавать спецификаторы форматов для выполнения локализации. Было создано пользовательское представление путем расширения класса SurfaceView. Вы узнали о том, что имена классов пользовательских компонентов должны быть полностью квалифицированы в элементе разметки XML, который представляет компонент.

Мы рассмотрели дополнительные методы «жизненного цикла» класса Activity, освоили вызов метода onPause из текущего класса Activity в случае, если другая деятельность становится активной и вызывается метод onDestroy, когда система завершает деятельность.

С помощью переопределения метода onTouchEvent класса Activity мы обрабатывали касания и одиночные тапы. Для обработки двойных тапов, которые вызывают выстрел из пушки, использовался GestureDetector. Событие двойного тапа вызывало метод SimpleOnGestureListener, который содержится в переопределенном методе onDoubleTap.

Мы добавили звуковые эффекты в папку приложения res/raw и управляли ими с помощью класса SoundPool, а также использовали системную службу AudioManager для получения установленного в устройстве текущего значения громкости звучания музыки и использования этого значения для установки громкости воспроизведения звука.

Это приложение вручную выполняет анимации путем обновления элементов игры на SurfaceView из отдельного потока выполнения. Был расширен класс Thread и использован метод run, с помощью которого отображается графика с привлечением методов класса Canvas. Мы использовали метод SurfaceHolder класса SurfaceView для получения соответствующего класса Canvas, а также научились создавать циклы, управляющие игрой на основе количества времени, которое прошло между кадрами анимации. В результате игра выполняется с одной и той же скоростью на различных устройствах.

В следующей главе мы создадим приложение SpotOn game — наше первое приложение Android 3.x. Приложение SpotOn использует свойство анимации Android 3.x для анимации компонентов View, включающих изображения. Это приложение применяется для тестирования реакции пользователя с помощью отображения нескольких кружочков, на которые нужно нажать до того, как они исчезнут с экрана.

8

Игра SpotOn

Анимация свойств, класс ViewPropertyAnimator, интерфейс AnimatorListener, потоково-безопасные коллекции, объекты SharedPreferences, заданные по умолчанию для деятельности

В этой главе...

- Создание игры, которую легко кодировать и в которую приятно играть.
- Использование класса ViewPropertyAnimator для группирования анимаций, которые перемещают и изменяют размеры компонентов ImageViews.
- Реагирование на события анимации «жизненного цикла» с помощью AnimatorListener.
- Обработка событий щелчков для компонентов ImageViews и событий касания экрана.
- Использование безопасной для потока коллекции ConcurrentLinkedQueue из пакета java.util.concurrent для обеспечения одновременного доступа к коллекции из нескольких потоков.

8.1. Введение

Игра SpotOn предназначена для проверки реакции пользователя — нужно успеть коснуться перемещающихся по экрану пятнышек до того, как они исчезнут (рис. 8.1). В процессе перемещения размеры пятнышек изменяются, это еще больше затрудняет задачу пользователя. Игра начинается с первого уровня. После касания 10 пятнышек осуществляется переход на следующий уровень. Чем больше номер уровня, тем выше

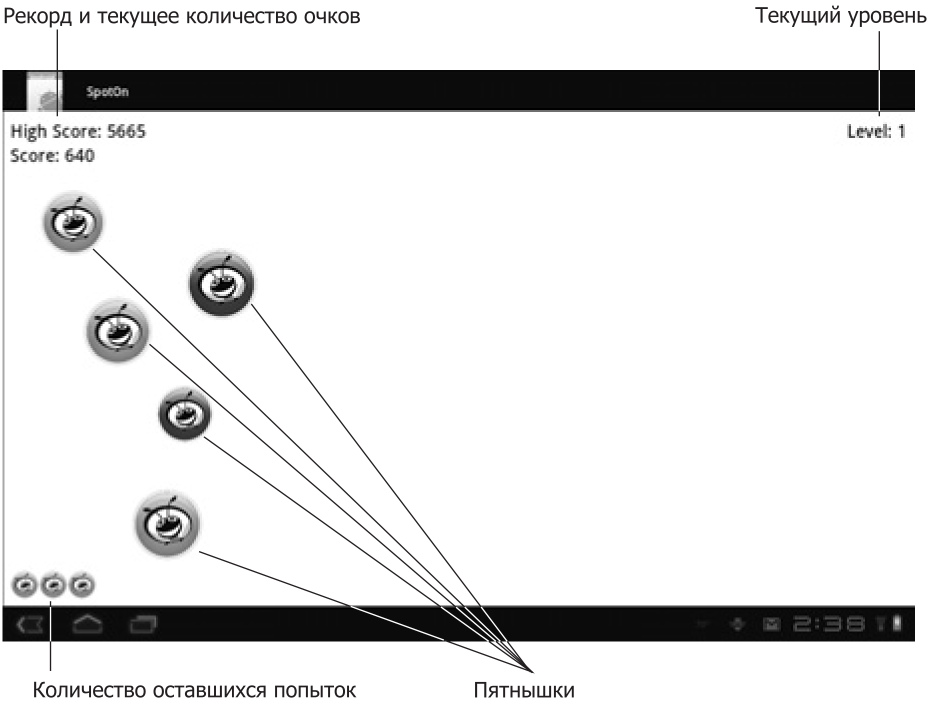


Рис. 8.1. Окно приложения SpotOn game

скорость перемещения пятнышек по экрану. После касания пятнышка раздается характерный «булькающий» звук, а само пятнышко исчезает с экрана. За каждое «пойманное» пятнышко начисляются призовые очки (при подсчете очков номер уровня умножается на 10). Будьте точны, ибо если вы не «попадете» по пятнышку, количество очков уменьшится на величину, равную числу 15 умноженному на номер уровня. В начале игры у каждого пользователя имеются *три* дополнительных попытки (жизни), отображенные в левом нижнем углу окна приложения. Если пятнышко исчезает до того, как его коснется пользователь, раздается характерный звук вытекающей жидкости, а количество дополнительных попыток уменьшается на одну. После перехода на новый уровень пользователь получает дополнительную попытку (до семи дополнительных попыток). Если дополнительные попытки отсутствует, а пользователь не успел коснуться ни одного пятнышка, игра завершается (рис. 8.2).

8.2. Тестирование приложения SpotOn Game

Открытие и запуск приложения

Откройте среду Eclipse и импортируйте проект SpotOn app. Выполните следующие действия:

1. *Отройте диалоговое окно* Import, для этого выполните команды File ▸ Import... (Файл ▸ Импорт...).

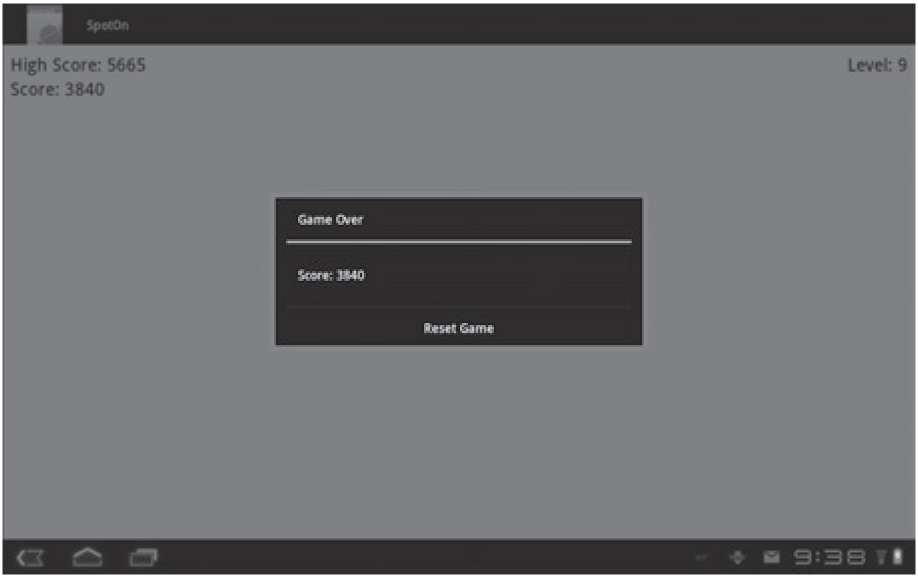


Рис. 8.2. Экран завершения игры, отображающий количество набранных очков и кнопку Reset Game

2. *Импортируйте проект приложения* SpotOn app. В диалоговом окне Import раскройте узел General (Общий) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область). Затем щелкните на кнопке Next > (Далее >) для выполнения шага Import Projects (Импорт проектов). Выберите параметр Select root directory (Выбрать корневой каталог) и щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папки) выберите папку SponOn в папке примеров книги и щелкните на кнопке OK. Щелкните на кнопке Finish (Готово) для импорта проекта в среду Eclipse. Проект отобразится в окне Package Explorer, находящемся в левой части окна Eclipse.
3. *Запустите приложение* SpotOn. В среде Eclipse щелкните правой кнопкой мыши на проекте SpotOn, находящемся в окне Package Explorer, затем в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android).

Процесс игры

После появления пятнышка на экране коснитесь его пальцем (или щелкните мышью в среде виртуального устройства AVD). Попытайтесь не упустить пятнышко, чтобы не потерять дополнительную жизнь. Игра завершится, как только будет исчерпано количество жизней и исчезнет пятнышко, которого вы не успеете коснуться.

ПРИМЕЧАНИЕ

Это приложение предназначено для операционной среды Android 3.1. На момент написания книги виртуальное устройство AVD для Android 3.0 и более поздней версии работало очень медленно. Поэтому лучше запускать это приложение на реальном устройстве Android 3.1.

8.3. Обзор применяемых технологий

Android 3.x и анимация свойств

Это первое приложение, использующее свойства Android 3.0+. Например, анимация свойств, появившаяся в версии Android 3.0, используется для перемещения и масштабирования компонентов ImageViews.

В версиях Android до 3.0 использовались два основных механизма анимации:

- *Анимации* Tweened View (анимация с переходами) позволяют изменять ограниченные аспекты видимости компонента View, например место отображения, вращение и размеры.
- *Анимации* Frame View (покадровая анимация) отображают последовательность изображений.

Если к анимации предъявляются особые требования, можно создать свою собственную анимацию (см. главу 7). К сожалению, анимации компонентов View влияют лишь на порядок рисования этих компонентов на экране. Если анимировать компонент Button так, что его изображение перемещается по экрану, пользователь может инициализировать событие щелчка на компоненте Button только путем касания его исходного местоположения на экране.

С помощью анимации свойства (package android.animation) можно анимировать произвольное свойство *любого* объекта (при этом механизм анимации не ограничен компонентами View). В результате перемещения компонента Button с помощью анимации свойства компонент Button не только рисуется в другом месте экрана, но и обеспечивается взаимодействие пользователя с компонентом Button в его текущем местоположении.

С помощью анимации свойств выполняется анимация *значений* на протяжении определенного времени. Для создания анимации укажите следующие параметры:

- целевой объект, включающий свойство (или свойства), которое будет анимироваться;
- анимируемое свойство (или свойства);
- длительность анимаций;
- значения, используемые при анимации, между свойствами;
- порядок изменения значений свойства в течение времени (*интерполятор*).

Классы анимации свойства — ValueAnimator и ObjectAnimator. Класс ValueAnimator применяется для вычисления значений свойства в течение определенного времени. При этом нужно определить метод AnimatorUpdateListener, с помощью которого программным образом изменяются значения свойства целевого объекта. Эта методика полезна в случае, если у целевого объекта отсутствует стандартный *набор* методов, применяемых для изменения значений свойства. Подкласс ValueAnimator класса ObjectAnimator использует *набор* методов целевого объекта для изменения свойств анимированного объекта в соответствии с изменением значений объекта, осуществляемым с течением времени.

В Android 3.1 появился новый класс утилит ViewPropertyAnimator, с помощью которого упрощается анимация свойств компонентов View и обеспечивается одновременная анимация нескольких свойств. Каждый компонент View включает метод анимации, возвращающий ViewPropertyAnimator, с помощью которого можно создать цепочку методов, вызываемых в процессе конфигурирования анимации. Как только завершится

выполнение последнего метода в цепочке, запускается анимация. Эта техника используется для анимации пятнышек в игре. Дополнительные сведения относительно анимации в Android можно найти в следующих блогах:

- android-developers.blogspot.com/2011/02/animation-in-honeycomb.html;
- android-developers.blogspot.com/2011/05/introducing-viewpropertyanimator.html.

Прослушивание событий «жизненного цикла» анимации

С помощью реализации интерфейса AnimatorListener можно прослушивать события «жизненного цикла» анимации свойства, которые определяют методы, вызываемые после запуска, завершения, повторения или отмены анимации. Если в приложении не требуется использование всех четырех видов анимаций, можно расширить класс AnimatorListenerAdapter и переопределить лишь требуемый(ые) метод слушателя.

Обработка касаний

В главе 7 вашему вниманию была предложена методика обработки прикосновений путем переопределения метода onTouchEvent класса Activity. В игре SpotOn используются два типа касаний — касание пятнышка и любого другого места экрана. Для обработки касаний пятнышек регистрируются OnClickListeners для каждого пятнышка (то есть ImageView), а для обработки других касаний экрана используется метод onTouchEvent.

Класс ConcurrentLinkedQueue и интерфейс Queue

С помощью класса ConcurrentLinkedQueue (из пакета java.util.concurrent) и интерфейса Queue поддерживаются безопасные для потока списки объектов, доступ к которым можно получить из нескольких потоков выполнения одновременно.

8.4. Создание графического интерфейса и файлов ресурсов приложения

В этом разделе будут созданы графический интерфейс пользователя и файлы ресурсов для приложения SpotOn game. В целях экономии места мы не отображаем файл ресурсов приложения strings.xml. Чтобы просмотреть содержимое этого файла, откройте его из проекта в среде Eclipse.

8.4.1. Файл AndroidManifest.xml

В листинге 8.1 показан файл приложения AndroidManifest.xml. Атрибуту uses-sdk элемента android:minSdkVersion присваивается значение "12" (строка 5), которое представляет Android 3.1 SDK. Это приложение будет выполняться только на устройствах Android 3.1+ и виртуальных устройствах AVD. В строке 7 атрибуту android:hardwareAccelerated присваивается значение true. Благодаря этому приложение может использовать *аппаратно ускоренную графику* (при наличии подобной возможности) для увеличения производительности. В строке 9 устанавливается значение атрибута android:screenOrientation, чтобы указать отображение приложения в *альбомной* (горизонтальной) *ориентации*.

Листинг 8.1. Файл AndroidManifest.xml

```
 <?xml version="1.0" encoding="utf-8"?>
 <manifest xmlns:android=http://schemas.android.com/apk/res/android
    android:versionCode="1" android:versionName="1.0"
    package="com.deitel.spoton">
    <uses-sdk android:minSdkVersion="12"/>
    <application android:icon="@drawable/icon"
    android:hardwareAccelerated="true" android:label="@string/app_name">
    <activity android:name=".SpotOn" android:label="@string/app_name"
       android:screenOrientation="landscape">
       <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER"/>
       </intent-filter>
    </activity>
  </application>
</manifest>
```

8.4.2. Файл main.xml RelativeLayout

Файл разметки main.xml приложения (листинг 8.2) содержит компонент RelativeLayout, позиционирующий компоненты приложения TextView, с помощью которых отображаются максимально набранное количество очков, текущий уровень и количество очков, а также компонент LinearLayout, используемый для отображения количества оставшихся попыток. Используемые в этом файле макеты и компоненты GUI были представлены ранее, поэтому здесь будут рассмотрены лишь ключевые свойства этого файла. Названия компонентов GUI приложения представлены на рис. 8.3.

Листинг 8.2. Файл разметки main.xml приложения SpotOn

```
 <?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android=http://schemas.android.com/apk/res/android
    android:id="@+id/relativeLayout" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/white">
    <TextView android:id="@+id/highScoreTextView"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginTop="10dp"
       android:layout_marginLeft="10dp"
       android:textColor="@android:color/black" android:textSize="25sp"
       android:text="@string/high_score"></TextView>
    <TextView android:id="@+id/levelTextView"
       android:layout_toRightOf="@id/highScoreTextView"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginTop="10dp"
       android:layout_marginRight="10dp"
       android:gravity="right"
       android:layout_alignParentRight="true"
       android:textColor="@android:color/black" android:textSize="25sp"
```

продолжение ☞

Листинг 8.2 *(продолжение)*

```
      android:text="@string/level"></TextView>
   <TextView android:id="@+id/scoreTextView"
      android:layout_below="@id/highScoreTextView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="10dp"
      android:textColor="@android:color/black" android:textSize="25sp"
      android:text="@string/score"></TextView>
   <LinearLayout android:id="@+id/lifeLinearLayout"
      android:layout_alignParentBottom="true"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_margin="10dp"></LinearLayout>
</RelativeLayout >
```

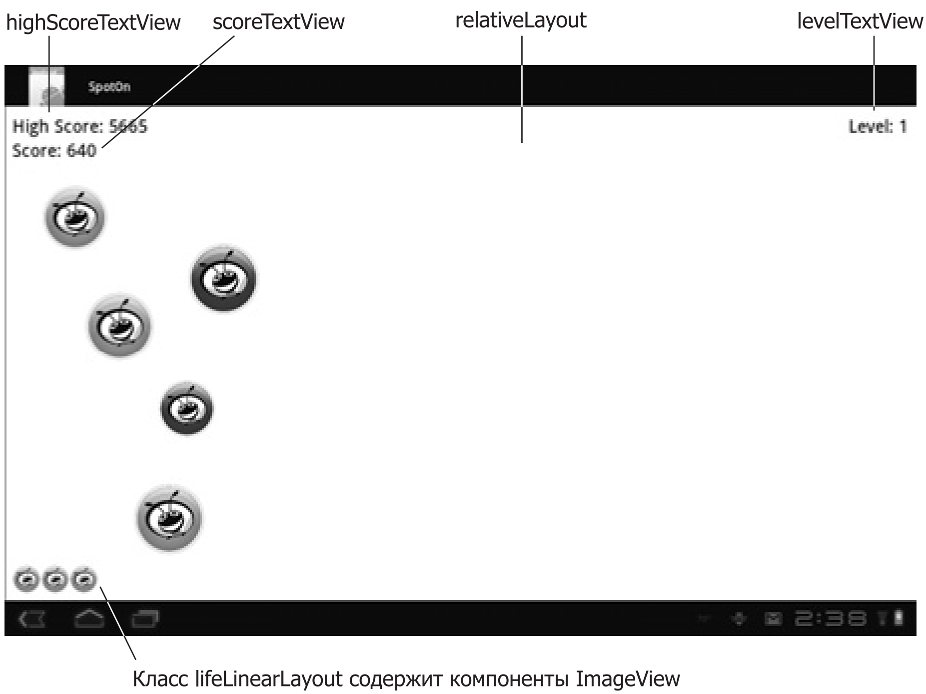


Рис. 8.3. Названия компонентов GUI приложения SpotOn

8.4.3. Файл разметки untouched.xml ImageView для нового пятнышка

Файл разметки приложения untouched.xml (листинг 8.3) определяет компонент ImageView, который может быть «раздут» и сконфигурирован динамически по мере создания каждого нового пятнышка в игре.

Листинг 8.3. Файл разметки untouched.xml ImageView приложения SpotOn для нового пятнышка

```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android="http://schemas.android.com/apk/res/android">
</ImageView>
```

8.4.4. Файл разметки life.xml ImageView для новых попыток

Файл разметки life.xml приложения (листинг 8.4) включает компонент ImageView, который может «раздуваться» и конфигурироваться динамически при добавлении новых попыток (жизней) в процессе игры.

Листинг 8.4. Файл разметки life.xml приложения SpotOn

```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android=http://schemas.android.com/apk/res/android
android:src="@drawable/life"></ImageView>
```

8.5. Создание приложения

Игра SpotOn состоит из двух классов — SpotOn (раздел 8.5.1), который является главным классом Activity приложения, и SpotOnView (раздел 8.5.2), определяющего логику игры и анимацию пятнышек.

8.5.1. Подкласс SpotOn класса Activity

Класс SpotOn (листинг 8.5) переопределяет метод onCreate, используемый для конфигурирования GUI. В строках 24–25 создается элемент SpotOnView, а в строке 26 происходит его добавление в компонент RelativeLayout (в позиции 0). В результате этот элемент помещается «позади» других элементов макета. Конструктор SpotOnView требует три аргумента — Context, в котором отображается компонент GUI (то есть класс Activity), объект SharedPreferences и объект RelativeLayout (в результате в данном макете обеспечивается взаимодействие SpotOnView с другими компонентами GUI). В главе 5 было продемонстрировано, каким образом происходит считывание и запись в файл SharedPreferences. В этом приложении по умолчанию используется один такой файл, он связан с классом Activity. Этот файл можно получить путем вызова метода getPreferences класса Activity.

Листинг 8.5. Класс SpotOn определяет главный класс Activity приложения

```
 1 // SpotOn.java
 2 // Класс Activity для приложения SpotOn
 3 package com.deitel.spoton;
 4
 5 import android.app.Activity;
 6 import android.content.Context;
 7 import android.os.Bundle;
 8 import android.widget.RelativeLayout;
 9
10 public class SpotOn extends Activity
```

продолжение ⇗

Листинг 8.5 *(продолжение)*

```
{
   private SpotOnView view; // вызывает игру и управляет ей

   // вызывается при первом создании Activity
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

   // создание нового компонента SpotOnView
   // и его добавление в RelativeLayout
   RelativeLayout layout =
   (RelativeLayout) findViewById(R.id.relativeLayout);
    view = new SpotOnView(this, getPreferences(Context.MODE_PRIVATE),
       layout);
    layout.addView(view, 0); // добавление представления в макет
}   // конец метода onCreate

// вызывается после перемещения Activity на фон
@Override
public void onPause()
{
   super.onPause();
   view.pause(); // освобождение ресурсов, выделенных для представления
}  // конец метода onPause

// вызывается при перемещении Activity на передний план
@Override
public void onResume()
{
   super.onResume();
   view.resume(this); // повторная инициализация ресурсов,
                      // освобожденных методом onPause
}  // конец метода onResume
}  // конец класса SpotOn
```

Переопределенные методы onPause и onResume класса Activity вызывают методы pause и resume компонента SpotOnView соответственно. Если вызывается метод onPause класса Activity, метод pause класса SpotOnView освобождает ресурсы SoundPool, используемые приложением, и отменяет все выполняющиеся приложения. Вы уже знаете, что после начала выполнения класса Activity вызывается его метод onCreate. Затем следуют последовательные вызовы методов onStart и onResume класса Activity. Метод onResume также вызывается в том случае, если Activity возвращается с фона на передний план. Если метод onResume вызывается в классе Activity приложения, метод resume класса SpotOnView снова получает ресурсы SoundPool и перезапускает игру. Если окно приложения не отображается на экране, состояние игры не сохраняется.

8.5.2. Подкласс SpotOnView класса View

Класс SpotOnView (листинги 8.6–8.18) определяет логику игры и анимацию пятнышек.

Операторы package и import

В разделе 8.3 были рассмотрены ключевые новые классы и интерфейсы, использующие класс SpotOnView. Все они приведены в листинге. 8.6.

Листинг 8.6. Операторы package и import класса SpotOnView

```
// SpotOnView.java
// Класс View, который отображает элементы и реализует управление игрой
package com.deitel.spoton;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.Queue;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.content.Context;
import android.content.DialogInterface;
import android.content.SharedPreferences;
import android.content.res.Resources;
import android.media.AudioManager;
import android.media.SoundPool;
import android.os.Handler;
import android.view.LayoutInflater;
import android.view.MotionEvent;
import android.view.View;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.RelativeLayout;
import android.widget.TextView;

```

Константы и переменные экземпляра класса

В листинге 8.7 начинается определение класса SpotOnView, а также объявляются константы и переменные экземпляра класса. В строках 33–34 определяется константа и переменная SharedPreferences, используемая для загрузки и хранения высших достижений (рекордов) игры в заданном по умолчанию файле SharedPreferences класса Activity. В строках 37–73 объявляются переменные и константы, применяемые для управления аспектами игры. Эти переменные будут подробно рассмотрены по мере их использования. В строках 76–84 определяются переменные и константы, применяемые

для управления и воспроизведения звуков игры. В главе 7 демонстрируется использование звуков в приложении.

Листинг 8.7. Константы и переменные экземпляра класса SpotOnView

```
public class SpotOnView extends View
{
// константа, применяемая для доступа к рекордным
// очкам в SharedPreference
private static final String HIGH_SCORE = "HIGH_SCORE";
private SharedPreferences preferences; // stores the high score

// переменные, применяемые для управления игрой
private int spotsTouched; // количество пятнышек, которых
                          // коснулся пользователь
private int score; // текущие очки
private int level; // текущий уровень
private int viewWidth; // хранится значение ширины представления
private int viewHeight; // хранится значение высоты представления
private long animationTime; // сколько каждое пятнышко
                            // остается на экране
private boolean gameOver;   // была ли завершена игра
private boolean gamePaused; // была ли завершена игра
private boolean dialogDisplayed; // была ли завершена игра
private int highScore; // рекордные очки за все время игры

// коллекции пятнышек (ImageViews) и Animators
private final Queue<ImageView> spots =
   new ConcurrentLinkedQueue<ImageView>();
private final Queue<Animator> animators =
   new ConcurrentLinkedQueue<Animator>();

private TextView highScoreTextView;    // отображает рекордные очки
private TextView currentScoreTextView; // отображает текущие очки
private TextView levelTextView; // отображает текущий уровень
private LinearLayout livesLinearLayout; // количество оставшихся
                                        // попыток
private RelativeLayout relativeLayout;  // отображает пятнышки
private Resources resources; // используется для загрузки ресурсов
private LayoutInflater layoutInflater;    // используется для
                                          // «раздувания» GUI

// время (в миллисекундах) анимации пятнышек и пятнышек,
// которых коснулся пользователь
private static final int INITIAL_ANIMATION_DURATION = 6000;
private static final Random random = new Random();  // для случайных
                                                    // координат
private static final int SPOT_DIAMETER = 100; // начальный
                                              // размер пятнышка
private static final float SCALE_X = 0.25f;    // конец анимации
                                              // по шкале x
```

```
private static final float SCALE_Y = 0.25f;   // конец анимации
                                              // по шкале y
private static final int INITIAL_SPOTS = 5;  // начальное количество
                                              // пятнышек
private static final int SPOT_DELAY = 500; // задержка в миллисекундах
private static final int LIVES = 3; // начало игры с тремя попытками
private static final int MAX_LIVES = 7;   // максимальное число попыток
private static final int NEW_LEVEL = 10;  // количество пятнышек,
                                          // требуемых для перехода
                                          // на следующий уровень
private Handler spotHandler; // добавление новых пятнышек в игре

// идентификаторы звука, константы и переменные для звуков игры
private static final int HIT_SOUND_ID = 1;
private static final int MISS_SOUND_ID = 2;
private static final int DISAPPEAR_SOUND_ID = 3;
private static final int SOUND_PRIORITY = 1;
private static final int SOUND_QUALITY = 100;
private static final int MAX_STREAMS = 4;
private SoundPool soundPool; // воспроизведение звуковых эффектов
private int volume; // громкость звукового эффекта
private Map<Integer, Integer> soundMap; // отображает ID на soundpool

```

Конструктор класса SpotOnView

Конструктор класса SpotOnView (см. листинг. 8.8) инициализирует несколько переменных экземпляра класса. В строке 93 хранится заданный по умолчанию объект SharedPreferences класса SpotOn, относящегося к Activity. Затем в строке 94 он применяется для загрузки рекордных очков. Второй аргумент указывает, что переменная getInt должна вернуть значение 0, если ключ HIGH_SCORE уже не существует. В строке 97 используется аргумент context для получения и хранения объекта Resources класса Activity. Мы используем этот объект для загрузки ресурсов String, применяемых для отображения текущих и рекордных очков, номер текущего уровня и финальные очки пользователя. В строках 100–101 хранится объект LayoutInflater, применяемый для динамического «раздувания» компонентов ImageView в игре. В строке 104 хранится ссылка на компонент RelativeLayout класса SpotOn класса Activity. В строках 105–112 этот компонент используется для получения ссылок на компонент LinearLayout, в котором отображаются дополнительные жизни, а также на компонент TextView, отображающий рекордные очки, текущие очки и номер уровня. В строке 114 создается объект Handler, который метод resetGame (см. листинг 8.11) использует для отображения первых нескольких пятнышек игры.

Листинг 8.8. Конструктор класса SpotOnView

```
86   // конструирует новый класс SpotOnView
87   public SpotOnView(Context context, SharedPreferences sharedPreferences,
88      RelativeLayout parentLayout)
89   {
90      super(context);
```

продолжение ⇨

Листинг 8.8 *(продолжение)*

```

// загрузка рекордных очков
preferences = sharedPreferences;
highScore = preferences.getInt(HIGH_SCORE, 0);

// сохранение ресурсов, используемых для загрузки внешних значений
resources = context.getResources();

// сохранение LayoutInflater
layoutInflater = (LayoutInflater) context.getSystemService(
Context.LAYOUT_INFLATER_SERVICE);

// получение ссылок на различные компоненты GUI
relativeLayout = parentLayout;
livesLinearLayout = (LinearLayout) relativeLayout.findViewById(
   R.id.lifeLinearLayout);
highScoreTextView = (TextView) relativeLayout.findViewById(
   R.id.highScoreTextView);
currentScoreTextView = (TextView) relativeLayout.findViewById(
   R.id.scoreTextView);
levelTextView = (TextView) relativeLayout.findViewById(
   R.id.levelTextView);

 spotHandler = new Handler(); // добавление пятнышек после запуска игры
} // конец конструктора SpotOnView

```

Переопределение метода onSizeChanged класса View

При вычислении случайных координат начальной и конечной точки каждого нового пятнышка используются значения ширины и высоты компонента SpotOnView. Размеры этого компонента не могут изменяться до тех пор, пока он не будет добавлен в иерархию объектов View. В результате вы не сможете изменять высоту и ширину этого компонента с помощью соответствующего конструктора. Кроме того, необходимо переопределить метод onSizeChanged класса View (см. листинг 8.9), чтобы иметь возможность его вызова сразу же после добавления класса View в иерархию объектов View и изменения размеров.

Листинг 8.9. Переопределение метода onSizeChanged класса View

```
// хранение значений width/height класса SpotOnView
@Override
protected void onSizeChanged(int width, int height,
   int oldw, int oldh)
{
   viewWidth = width;   // сохранение нового значения ширины
   viewHeight = height; // сохранение нового значения высоты
}  // завершение метода onSizeChanged

```

Методы pause, cancelAnimations и resume

Методы pause, cancelAnimations и resume (см. листинг 8.10) облегчают управление ресурсами приложения и позволяют завершить анимацию в случае, если приложение не отображается на экране.

Листинг 8.10. Методы pause, cancelAnimations и resume класса SpotOnView

```
125   // Вызывается класс SpotOn из Activity при приеме вызова onPause
126   public void pause()
127   {
128      gamePaused = true;
129      soundPool.release(); // освобождение аудиоресурсов
130      soundPool = null;
131      cancelAnimations();  // отменяются все невыполненные анимации
132   }  // конец метода pause
133
134   // отмена анимаций и удаление ImageViews, представляющих пятнышки
135   private void cancelAnimations()
136   {
137      // отмена оставшихся анимаций
138      for (Animator animator : animators)
139         animator.cancel();
140
141   // удаление оставшихся пятнышек с экрана
142   for (ImageView view : spots)
143      relativeLayout.removeView(view);
144
145      spotHandler.removeCallbacks(addSpotRunnable);
146      animators.clear();
147      spots.clear();
148   }  // завершение метода cancelAnimations
149
150   // вызывается классом SpotOn из Activity при приеме вызова
      // метода onResume
151   public void resume(Context context)
152   {
153      gamePaused = false;
154      initializeSoundEffects(context); // инициализация SoundPool
                                         // приложения
155
156      if (!dialogDisplayed)
157         resetGame(); // запуск игры
158   }  // завершение метода resume
159
```

После вызова метода onPause класса Activity вызывается метод pause (строки 126–132), который освобождает ресурсы SoundPool, используемые приложением, а затем вызывается метод cancelAnimations. Переменная gamePaused (см. листинг 8.15) применяется для исключения вероятности вызова метода missedSpot после завершения анимации и если приложение не отображается на экране.

Метод cancelAnimations (строки 135–148) выполняет итерации по коллекции аниматоров и вызывает метод cancel для каждого объекта Animator. В результате тут же завершается выполнение каждой анимации и вызываются методы onAnimationCancel и onAnimationEnd класса AnimationListener.

После вызова метода onResume класса Activity метод resume (строки 151–158) снова получает доступ к ресурсам SoundPool путем вызова initalizeSoundEffects (см. листинг 8.12). Если значение переменной dialogDisplayed равно true, диалоговое окно завершения игры по-прежнему отображается на экране. Чтобы начать новую игру, следует щелкнуть на кнопке Reset Game (Переустановить игру), находящейся в диалоговом окне. Если же это диалоговое окно не отображается, в строке 157 вызывает метод resetGame (см. листинг 8.11), запускающий новую игру.

Метод resetGame

Метод resetGame (см. листинг 8.11) восстанавливает игру в ее начальном состоянии, отображает начальные дополнительные жизни (попытки) и составляет график отображения начальных пятнышек. В строках 163–164 очищаются коллекции пятнышек и аниматоров, а в строке 165 используется метод removeAllViews класса ViewGroup для удаления компонентов ImageView, соответствующих дополнительным жизням, из компонента livesLinearLayout. В строках 167–171 переустанавливаются переменные экземпляра класса, используемые для управления игрой:

- переменная animationTime определяет длительность каждой анимации — для каждого нового уровня время анимации уменьшается на 5 % по сравнению с предыдущим уровнем;
- с помощью переменной spotsTouched облегчается определение момента перехода на новый уровень в случае достижения количества пятнышек, определенных переменной NEW_LEVEL;
- переменная score хранит текущее количество очков;
- в переменной level хранится номер текущего уровня;
- с помощью переменной gameOver определяется, завершена ли игра.

Листинг 8.11. Метод resetGame класса SpotOnView

```
160   // начало новой игры
161   public void resetGame()
162   {
163   spots.clear();     // очистка списка пятнышек
164   animators.clear(); // очистка списка аниматоров
165   livesLinearLayout.removeAllViews(); // очистка старых жизней с экрана
166
167   animationTime = INITIAL_ANIMATION_DURATION; // инициализация
                                                  // длины анимации
168   spotsTouched = 0; // переустановка числа пятнышек, которых
                        // коснулся пользователь
169   score = 0; // переустановка очков
170   level = 1; // переустановка уровня
171   gameOver = false; // игра не завершена
```

```
172   displayScores();  // отображение очков и уровня
173
174   // добавление жизней
175   for (int i = 0; i < LIVES; i++)
176   {
177      // добавление индикатора жизни на экран
178      livesLinearLayout.addView(
179         (ImageView) layoutInflater.inflate(R.layout.life, null));
180   }  // конец цикла for
181
182   // добавление новых пятнышек INITIAL_SPOTS в интервалы времени
      // SPOT_DELAY, выраженных в миллисекундах
183   for (int i = 1; i <= INITIAL_SPOTS; ++i)
184      spotHandler.postDelayed(addSpotRunnable, i * SPOT_DELAY);
185   }  // конец метода resetGame
186
```

В строке 172 вызывается метод displayScores (см. листинг 8.13), выполняющий переустановку компонентов TextView игры. В строках 175–180 выполняется повторяющееся «раздувание» файла life.xml с добавлением в макет livesLinearLayout каждого нового созданного компонента ImageView. И наконец, в строках 183–184 с помощью обработчика spotHandler планируется отображение первых нескольких пятнышек игры по истечении интервалов времени, заданных SPOT_DELAY (в миллисекундах).

Method initializeSoundEffects

Метод initializeSoundEffects (см. листинг 8.12) использует методики, которые применялись при разработке приложения Cannon Game (раздел 7.5.3), для создания звуковых эффектов игры. В этой игре используются три звука, представленные следующими ресурсами:

- R.raw.hit воспроизводится после касания пальцем пятнышка;
- R.raw.miss воспроизводится, если пальцем коснуться экрана, но не попасть в пятнышко;
- R.raw.disappear воспроизводится в случае, если завершена анимация пятнышка и его не коснулся пользователь.

Упомянутые выше звуковые MP3-файлы можно найти в папке примеров книги.

Листинг 8.12. Метод initializeSoundEffects класса SpotOnView

```
// создание класса SoundPool для воспроизведения звука в игре
private void initializeSoundEffects(Context context)
{
 // инициализация SoundPool для воспроизведения трех
 // звуковых эффектов, используемых в приложении
soundPool = new SoundPool(MAX_STREAMS, AudioManager.STREAM_MUSIC,
   SOUND_QUALITY);

// настройка громкости звуковых эффектов
AudioManager manager =
```

продолжение ⇗

Листинг 8.12 *(продолжение)*

```
      (AudioManager) context.getSystemService(Context.AUDIO_SERVICE);
   volume = manager.getStreamVolume(AudioManager.STREAM_MUSIC);

   // создание карты звука
   soundMap = new HashMap<Integer, Integer>(); // создание новой HashMap

   // добавление каждого звукового эффекта в SoundPool
   soundMap.put(HIT_SOUND_ID,
      soundPool.load(context, R.raw.hit, SOUND_PRIORITY));
   soundMap.put(MISS_SOUND_ID,
      soundPool.load(context, R.raw.miss, SOUND_PRIORITY));
   soundMap.put(DISAPPEAR_SOUND_ID,
      soundPool.load(context, R.raw.disappear, SOUND_PRIORITY));
   }  // конец метода initializeSoundEffect

```

Метод displayScores

Метод displayScores (см. листинг 8.13) просто обновляет три компонента TextView игры рекордными очками, текущими очками и номером текущего уровня. Части каждой строки загружаются файлом strings.xml с помощью метода getString объекта resources.

Листинг 8.13. Метод displayScores класса SpotOnView

```
   // отображение очков и номера уровня
   private void displayScores()
   {
      // отображение рекордных очков, текущих очков и номера уровня
      highScoreTextView.setText(
         resources.getString(R.string.high_score) + " " + highScore);
      currentScoreTextView.setText(
         resources.getString(R.string.score) + " " + score);
      levelTextView.setText(
         resources.getString(R.string.level) + " " + level);
   }   // конец функции displayScores

```

Класс Runnable AddSpotRunnable

Если метод resetGame (см. листинг 8.14) использует обработчик spotHandler для планирования графика отображения начальных пятнышек игры, при вызове метода postDelayed класса spotHandler в качестве аргумента принимается addSpotRunnable (см. листинг 8.14). Метод run класса Runnable просто вызывает метод addNewSpot (см. листинг 8.15).

Листинг 8.14. С помощью класса Runnable addSpotRunnable в игру добавляется новое пятнышко

```
   // Класс Runnable применяется для добавления новых пятнышек
   // в начале игры
```

```
private Runnable addSpotRunnable = new Runnable()
{
   public void run()
   {
      addNewSpot(); // добавление нового пятнышка в игру
   }  // конец метода run
}; // конец Runnable

```

Метод addNewSpot

Метод addNewSpot (см. листинг 8.15) добавляет новое пятнышко на экран игры. Этот метод вызывается несколько раз в начале игры для отображения начальных пятнышек, в случае если пользователь коснулся пятнышка (пятнышек) или же если пользователь не коснулся ни одного пятнышка, а анимация была завершена.

В строках 236–239 используются значения ширины и высоты компонента SpotOnView для выбора случайных координат для начальной и конечной точек анимации пятнышка. Затем в строках 242–250 «раздувается» и конфигурируется компонент ImageView для нового пятнышка. В строках 245–246 определяется ширина и высота компонента ImageView путем вызова его метода setLayoutParams с новым объектом RelativeLayout.LayoutParams. Затем в строках 247–248 производится случайный выбор между двумя ресурсами изображения и вызовом метода setImageResource класса ImageView для выбора изображения пятнышка. В строках 249–250 устанавливается начальное положение пятнышка. В строках 251–259 настраивается слушатель OnClickListener класса ImageView для вызова метода touchedSpot (см. листинг 8.17) после касания пальцем компонента ImageView. Затем добавляется пятнышко в макет relativeLayout, который отображает его на экране.

Листинг 8.15. Метод addNewSpot класса SpotOnView

```
232   // добавляет новое пятнышко в случайное местоположение
      // и начинает анимацию
233   public void addNewSpot()
234   {
235      // выбор двух случайных координат для начальной и конечной точек
236      int x = random.nextInt(viewWidth – SPOT_DIAMETER);
237      int y = random.nextInt(viewHeight – SPOT_DIAMETER);
238      int x2 = random.nextInt(viewWidth – SPOT_DIAMETER);
239      int y2 = random.nextInt(viewHeight – SPOT_DIAMETER);
240
241      // создание нового пятнышка
242      final ImageView spot =
243         (ImageView) layoutInflater.inflate(R.layout.untouched, null);
244      spots.add(spot); // добавление нового пятнышка в список пятнышек
245      spot.setLayoutParams(new RelativeLayout.LayoutParams(
246         SPOT_DIAMETER, SPOT_DIAMETER));
247      spot.setImageResource(random.nextInt(2) == 0 ?
248      R.drawable.green_spot : R.drawable.red_spot);
249      spot.setX(x); // выбор начальной координаты x пятнышка
250      spot.setY(y); // выбор начальной координаты y пятнышка
```

продолжение ☞

Листинг 8.15 *(продолжение)*

```
251      spot.setOnClickListener( // прослушивание пятнышек,
                                  // на которых щелкнули
252      new OnClickListener()
253      {
254         public void onClick(View v)
255         {
256            touchedSpot(spot); // обработка пятнышка, которого
                                  // коснулись пальцем
257         }  //  конец метода onClick
258      }  // конец OnClickListener
259   ); // завершение вызова setOnClickListener
260   relativeLayout.addView(spot); // добавление пятнышка на экран
261
262   // настройка и запуск анимации пятнышка
263   spot.animate().x(x2).y(y2).scaleX(SCALE_X).scaleY(SCALE_Y)
264   setDuration(animationTime).setListener(
265   new AnimatorListenerAdapter()
266   {
267      @Override
268      public void onAnimationStart(Animator animation)
269      {
270         animators.add(animation); // сохранить для возможной отмены
271      }  // завершение метода onAnimationStart
272
273      public void onAnimationEnd(Animator animation)
274      {
275         animators.remove(animation); // анимация завершена, удаление
276
277         if (!gamePaused && spots.contains(spot)) // нет касания
278         {
279            missedSpot(spot); // потеряна жизнь
280         }  // конец блока if
281      }  // конец метода onAnimationEnd
282   }  // конец AnimatorListenerAdapter
283   ); // завершение вызова setListener
284   } // конец метода addNewSpot
285
```

В строках 263–283 настраивается аниматор пятнышка ViewPropertyAnimator, который возвращается с помощью метода animate компонента View. Аниматор ViewPropertyAnimator конфигурирует анимации для чаще всего анимируемых свойств компонента View — альфа (прозрачность), вращение, масштабирование, трансляция (перемещение относительно текущей позиции) и местоположение. Дополнительно аниматор ViewPropertyAnimator поддерживает методы, используемые для настройки длительности анимации, AnimatorListener (для ответа на события «жизненного цикла» анимации) и интерполятор TimeInterpolator (для определения порядка вычисления значений свойства во время анимации). Чтобы сконфигурировать анимацию, вызовите «цепочку» методов ViewPropertyAnimator. В настоящем примере используются следующие методы:

- X — определяет заключительное значение координаты *x* компонента View.
- Y — определяет заключительное значение координаты *y* компонента View.

- scaleX — определяет заключительное значение ширины компонента View (в виде процента от исходной ширины).
- scaleY — определяет заключительное значение высоты компонента View (в виде процента от исходной высоты).
- setDuration — определяет длительность анимации (в миллисекундах).
- setListener — определяет слушатель анимации AnimatorListener.

Как только будет вызван последний метод в цепочке (в рассматриваемом случае это setListener), анимация запускается. Если не указан TimeInterpolator, по умолчанию используется LinearInterpolator. При этом значения свойств в процессе выполнения анимации не изменяются. Список заранее определенных интерполяторов можно найти на веб-сайте по адресу developer.android.com/reference/android/animation/TimeInterpolator.html.

Для слушателя AnimatorListener создан анонимный класс, который расширяет класс AnimatorListenerAdapter, поддерживающий пустые определения методов для каждого из четырех методов AnimatorListener. В данном случае были переопределены методы onAnimationStart и onAnimationEnd.

После начала выполнения анимации вызывается метод слушателя onAnimationStart. Этот метод в качестве аргумента получает Animator, который поддерживает методы манипулирования выполняющейся анимацией. Объект Animator хранится в коллекции аниматоров. После вызова метода onPause класса SpotOn Activity объекты Animator из коллекции применяются для прекращения выполнения анимаций.

После завершения выполнения анимации вызывается метод слушателя onAnimationEnd. Соответствующий объект Animator удаляется из коллекции animators (по причине его ненадобности). Если игра не приостановлена и пятнышко остается в коллекции пятнышек, вызывается метод missedSpot (см. листинг 8.18), с помощью которого определяется, что пользователь не коснулся пятнышка и одна попытка утеряна. Если пользователь коснулся пятнышка, оно исключается из коллекции пятнышек.

Переопределение метода onTouchEvent класса View

Переопределенный метод onTouchEvent класса View (см. листинг 8.16) «отвечает» на касания пальцем экрана, при которых «пропускаются» пятнышки. При этом воспроизводится звук «пропущенного» пятнышка, из суммы очков вычитается значение, полученное умножением номера уровня на 15. При этом проверяется, чтобы сумма очков была большей либо равной нулю, и отображается новая сумма очков.

Листинг 8.16. Переопределение метода onTouchEvent класса View

```
// вызывается, если пользователь касается экрана, но не пятнышка
@Override
public boolean onTouchEvent(MotionEvent event)
{
   // звук «пропущенного» пятнышка
   if (soundPool != null)
      soundPool.play(MISS_SOUND_ID, volume, volume,
         SOUND_PRIORITY, 0, 1f);
```

продолжение ⇗

Листинг 8.16 *(продолжение)*

```

      score -= 15 * level; // удаление некоторых точек
      score = Math.max(score, 0); // не допускаются отрицательные очки
      displayScores(); // обновление очков/номер уровня на экране
      return true;
   } // конец метода onTouchEvent

```

Метод touchedSpot

Метод touchedSpot (см. листинг 8.17) вызывается после каждого касания компонента ImageView, представляющего пятнышко. При этом удаляется пятнышко из игры, обновляется сумма очков и воспроизводится звук «попадания» в область пятнышка. Затем определяется, достиг ли пользователь следующего уровня и нужно ли добавить дополнительную попытку (жизнь), отображенную на экране (дополнительная попытка добавляется только в том случае, если не достигнуто максимальное количество попыток). И наконец, отображается обновленная сумма очков, и, если игра не завершена, на экране появляется новое пятнышко.

Листинг 8.17. Метод touchedSpot класса SpotOnView

```
   // вызывается при касании пятнышка
   private void touchedSpot(ImageView spot)
   {
      relativeLayout.removeView(spot); // после касания
                          // пятнышка оно удаляется с экрана
      spots.remove(spot); // старое пятнышко удаляется из списка

      ++spotsTouched; // увеличивается на единицу количество
                      // «затронутых» пятнышек
      score += 10 * level; // увеличение количества очков

      // воспроизведение звука «попадания»
      if (soundPool != null)
         soundPool.play(HIT_SOUND_ID, volume, volume,
            SOUND_PRIORITY, 0, 1f);

      // переход на следующий уровень после касания 10 пятнышек
      // на текущем уровне
      if (spotsTouched % 10 == 0)
      {
         ++level; // увеличение на единицу номера уровня
         animationTime *= 0.95; // скорость игры возрастает на 5%
                                // по сравнению с предыдущим уровнем

      // если не достигнуто максимальное число попыток
      if (livesLinearLayout.getChildCount() < MAX_LIVES)
      {
         ImageView life =
            (ImageView) layoutInflater.inflate(R.layout.life, null);
```

```
         livesLinearLayout.addView(life); // добавление дополнительной
                                          // попытки на экран
      }  // конец блока if
   }  // конец блока if

   displayScores(); // обновление суммы очков/номера уровня на экране

   if (!gameOver)
      addNewSpot(); // добавление другого «нетронутого» пятнышка
   }  // конец метода touchedSpot

```

Method missedSpot

Метод missedSpot (см. листинг 8.18) вызывается в том случае, если во время анимации пятнышка его не коснулся пользователь. Это пятнышко исключается из игры и, если игра завершена, происходит немедленный выход из метода. В противном случае воспроизводится звук «пропадающего» пятнышка. Затем определяется, должна ли игра завершиться. Если игра должна быть завершена, проверяется, было ли достигнуто рекордное количество очков и если это так, значение сохраняется (строки 356–362). Затем прекращаются все выполняющиеся анимации и отображается диалоговое окно, в котором выводится набранная пользователем сумма очков. Если у пользователя остались попытки (жизни), в строках 385–390 удаляется одна попытка и добавляется новое пятнышко.

Листинг 8.18. Метод missedSpot класса SpotOnView

```
336   // вызывается, если пользователь не коснулся пятнышка
      // во время анимации
337   public void missedSpot(ImageView spot)
338   {
339      spots.remove(spot); // удаление пятнышка из списка пятнышек
340      relativeLayout.removeView(spot); // удаление пятнышка с экрана
341
342      if (gameOver) // если игра завершена, выйти
343         return;
344
345      // воспроизведение звука «пропадающего» пятнышка
346      if (soundPool != null)
347         soundPool.play(DISAPPEAR_SOUND_ID, volume, volume,
348            SOUND_PRIORITY, 0, 1f);
349
350      // если игра проиграна
351      if (livesLinearLayout.getChildCount() == 0)
352      {
353         gameOver = true; // игра завершена
354
355      // если сумма очков больше рекордной суммы очков
356      if (score > highScore)
357      {
358         SharedPreferences.Editor editor = preferences.edit();
```

продолжение ⇗

Листинг 8.18 *(продолжение)*

```
            editor.putInt(HIGH_SCORE, score);
            editor.commit(); // сохранение новой рекордной суммы очков
            highScore = score;
         }  // конец блока if

         cancelAnimations();

         // отображение диалогового окна рекордной суммы очков
         Builder dialogBuilder = new AlertDialog.Builder(getContext());
         dialogBuilder.setTitle(R.string.game_over);
         dialogBuilder.setMessage(resources.getString(R.string.score) +
            " " + score);
         dialogBuilder.setPositiveButton(R.string.reset_game,
            new DialogInterface.OnClickListener()
            {
               public void onClick(DialogInterface dialog, int which)
               {
                  displayScores(); // проверка обновления суммы очков
                  dialogDisplayed = false;
                  resetGame(); // начало новой игры
               }  // конец метода onClick
            }  // завершение DialogInterface
         ); // завершение вызова dialogBuilder.setPositiveButton
         dialogDisplayed = true;
         dialogBuilder.show(); // отображение окна переустановки игры
      }  // конец блока if
         else // удаление одной попытки
         {
            livesLinearLayout.removeViewAt( // удаление попытки с экрана
            livesLinearLayout.getChildCount() — 1);
            addNewSpot(); // добавление другого пятнышка
         }  // конец блока else
     } // конец метода missedSpot
   } // конец класса SpotOnView
```

8.6. Резюме

В этой главе вашему вниманию была представлена игра SpotOn, предназначенная для проверки скорости реакции пользователя, которому нужно успеть коснуться перемещающихся по экрану пятнышек прежде, чем они исчезнут. Это первое в этой книге приложение, в котором используются функции, специфичные для Android 3.0 и более поздней версии. В частности, для перемещения и масштабирования компонентов ImageView использовалась анимация свойств, которая появилась в Android 3.0.

Вы узнали, что в версиях Android, предшествующих 3.0, применялись два механизма анимации. Анимация компонентов View с переходами, позволяющая изменять ограниченный набор аспектов отображения компонентов View, а также покадровая анимация компонентов View, предусматривающая отображение последовательности изображений.

Вы также узнали, что анимации компонентов View влияют лишь на порядок рисования компонентов View на экране.

Затем вашему вниманию были представлены анимации свойств, использующиеся для анимации любого свойства объекта. Анимации свойств могут анимировать значения на протяжении определенного времени. Необходимые условия подобной анимации: целевой объект, включающий анимируемое свойство (свойства), длительность анимации, значения анимации для каждого свойства и порядок изменения значений свойства с течением времени.

Были рассмотрены классы Android 3.0 ValueAnimator и ObjectAnimator, также был рассмотрен новый класс утилиты Android 3.1 ViewPropertyAnimator, который был добавлен в библиотеки API анимации в целях упрощения анимации свойств для компонентов View и обеспечения возможности одновременной анимации (параллельный режим).

Мы воспользовались методом animate компонента View для получения подкласса ViewPropertyAnimator класса View, затем построили цепочку вызовов методов, применяемую для настройки анимации. После завершения выполнения последнего метода в цепочке начинает выполняться анимация. События «жизненного цикла» анимации свойств «прослушиваются» с помощью реализации интерфейса AnimatorUpdateListener, который определяет методы, вызываемые во время запуска, завершения, повторения или отмены анимации. И поскольку нам требуются лишь два события «жизненного цикла», мы реализовали слушатель путем расширения класса AnimatorListenerAdapter.

И наконец, мы использовали класс ConcurrentLinkedQueue из пакета java.util.concurrent и интерфейс Queue для поддержки безопасного для потока списка объектов, доступ к которому можно получить из нескольких потоков выполнения одновременно.

В главе 9 рассматривается разработка приложения Doodlz, которое использует графические возможности Android, позволяющие превратить экран устройства в *виртуальный холст*.

9

Приложение Doodlz

Двумерная графика, диспетчер SensorManager, мультитач-события и объекты Toast

В этой главе...

- Обнаружение событий, связанных с касанием экрана, перемещением пальца вдоль экрана и убирания пальца с экрана.
- Обработка нескольких касаний экрана, позволяющая рисовать несколькими пальцами одновременно.
- Использование диспетчера SensorManager для обнаружения событий, связанных с перемещением акселерометра, для очистки экрана при встряхивании устройства.
- Использование булевой переменной AtomicBoolean, к которой обеспечивается доступ из нескольких потоков в режиме безопасности потоков.
- Применение объекта Paint для определения цвета и ширины линии.
- Использование объектов Path для хранения данных каждой линии при рисовании линий пользователем и при создании этих линий с помощью объекта Canvas.
- Отображение кратких сообщений на экране с помощью объекта Toast.

9.1. Введение

С помощью приложения Doodlz экран вашего устройства превращается в *виртуальный холст* (рис. 9.1). Рисование на этом холсте осуществляется перемещением одного или нескольких пальцев вдоль экрана. Параметры приложения позволяют настроить

цвет и толщину линии. В диалоговом окне Choose Color (Выбор цвета), показанном на рис. 9.2, *а*, отображаются компоненты SeekBar (ползунки), с помощью которых настраиваются интенсивность альфа-канала (прозрачность), красного, зеленого и синего цветов, позволяя выбрать цвет ARGB. По мере перемещения ползунка каждого компонента SeekBar в образце цвета, отображенном ниже компонентов SeekBar, отображается текущий цвет. В диалоговом окне Choose Line Width (Выбор ширины линии), показанном на рис. 9.2, *б*, отображается единственный компонент SeekBar, с помощью которого настраивается толщина рисуемой линии. Дополнительные параметры меню (рис. 9.3) позволяют превратить ваш палец в ластик (Erase), очистить экран (Clear) или сохранить текущий рисунок в галерее устройства (Save Image). Чтобы полностью очистить экран от рисунков, просто встряхните устройство.



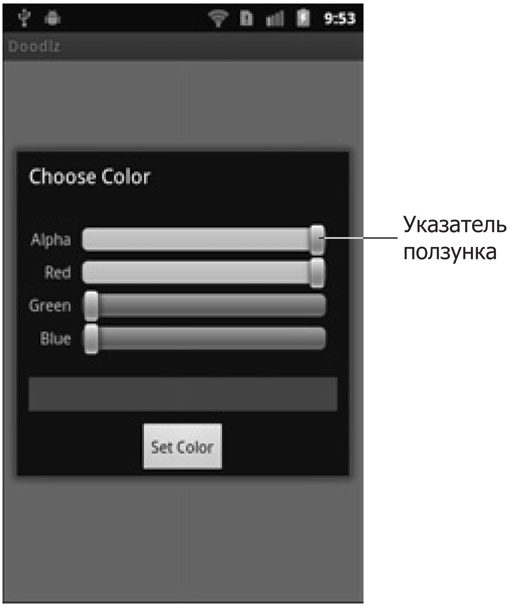
Рис. 9.1. Окно приложения Doodlz с завершенным рисунком

9.2. Тестирование приложения Doodlz

Тестирование этого приложения выполнялось в разделе 1.11 главы 1, поэтому в данный момент выполняться не будет.

9.3. Обзор применяемых технологий

В этом разделе мы рассмотрим множество новых технологий, используемых при создании приложения Doodlz в том порядке, в котором они будут встречаться в главе.





а *б*

Рис. 9.2. Диалоговые окна приложения Doodlz: *а* — диалоговое окно Choose Color; *б* — диалоговое окно Choose Line Width

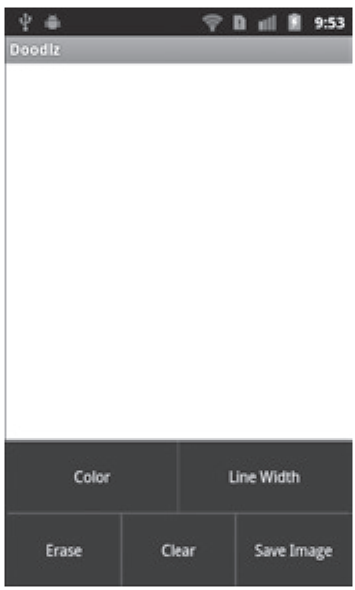


Рис. 9.3. Параметры меню приложения Doodlz

Обеспечение лучшей интеграции приложения с Android версии 3.0 либо более поздней

Несмотря на то что в приложении не используются функции Android 3.0, в файле манифеста приложения указывается библиотека Android 3.0 SDK (раздел 9.4.2). Это дает возможность использования в компонентах GUI приложения свойств интерфейса Android 3.0, образно называемых «смотри и ощущай» (так называемая *голографическая тема*). В результате интерфейс вашего приложения будет выглядеть просто шикарно на экранах планшетных устройств Android. Кроме того, в правой части панели действий Android 3.0, находящейся в верхней части экрана планшета, будет отображаться меню приложения.

Использование SensorManager для прослушивания событий акселерометра

Для удаления текущего рисунка, созданного с помощью приложения, достаточно встряхнуть устройство. Большинство устройств Android снабжены акселерометрами, с помощью которых приложения отслеживают перемещения устройств. Также в настоящее время Android поддерживает сенсоры гравитации, гироскопы, сенсоры освещения, линейного ускорения, магнитного поля, давления, приближения, вектора поворота и температуры. Перечень сенсоров, включающий описанные типы сенсоров, вы найдете на веб-сайте по адресу developer.android.com/reference/android/hardware/Sensor.html.

Для прослушивания события сенсоров получите ссылку на системную службу SensorManager (раздел 9.5.1), обеспечивающую получение данных приложением от сенсоров устройства. Воспользуйтесь службой SensorManager для регистрации изменений состояния сенсоров, которые должно обрабатывать приложение, а также укажите обработчик SensorEventListener для обработки событий, связанных с изменением состояния сенсоров. Классы и интерфейсы, применяемые для обработки событий сенсоров, находятся в пакете android.hardware.

Создание пользовательских диалоговых окон

В приложениях, создаваемых в предыдущих главах книги, использовались диалоговые окна AlertDialog для отображения информации пользователю или для отображения вопросов и получения ответов от пользователя в виде щелчков на компонентах Button. В диалоговых окнах AlertDialog могут отображаться только простые строки String и кнопки Button. В более сложных диалоговых окнах можно использовать объекты класса Dialog (пакет android.app), отображающего пользовательские графические интерфейсы (раздел 9.5.1). В разрабатываемом в данной главе приложении благодаря этим объектам пользователь может выбирать цвет рисунка или толщину линии. Графический интерфейс для каждого класса Dialog был получен путем «раздувания» файла XML-разметки (листинги 9.4 и 9.5).

Объект AtomicBoolean

В Android события, порожденные сенсорами, обрабатываются в отдельном потоке выполнения, предназначенному для событий GUI. В процессе обработки события, порожденного встряхиванием корпуса устройства, обработчик может попытаться отобразить

диалоговое окно подтверждения, в котором предлагается удалить изображение, если на экране появляется другое диалоговое окно. Чтобы исключить подобное поведение, используется объект AtomicBoolean (пакет import java.util.concurrent.atomic), с помощью которого определяется, отображается ли в настоящее время диалоговое окно. Объект AtomicBoolean управляет булевым значением потокобезопасным способом. В результате это значение становится доступным из нескольких потоков выполнения. Если значение AtomicBoolean равно true, мы запрещаем обработчику события встряхивания отображать диалоговое окно.

Пользовательские цвета

Пользователь может настроить заказной компонент Color, используемый при рисовании (раздел 9.5.1) в приложении. При этом определяются компоненты Color: альфа, красный, зеленый и синий вместе с ползунками SeekBar в диалоговом окне Dialog. Каждое значение находится в диапазоне от 0 до 255. Компонент альфа определяет прозрачность компонента Color, причем 0 представляет полностью прозрачный цвет, а 255 — полностью непрозрачный цвет. Класс Color поддерживает методы, обеспечивающие «сборку» Color на основе значений его компонентов (это необходимо в случае выбора пользовательского цвета). Также методы этого класса применяются для получения значений компонентов на основе Color (при необходимости установки начальных значений ползунков SeekBar в диалоговом окне Choose Color).

Рисование линий и контуров

Это приложение рисует линии на Bitmap (пакет android.graphics). Свяжите объект Canvas с Bitmap, затем с помощью Canvas начните рисовать на Bitmap, который будет отображаться на экране (разделы 9.5.1 и 9.5.2). Объект Bitmap также можно сохранить в файле. Эта возможность используется для сохранения рисунков в галерее устройства после выбора пользователем команды меню Save Image (Сохранить изображение).

Обработка событий касания экрана

Пользователь может коснуться экрана одним или несколькими пальцами либо выполнить перетаскивание пальцами для рисования линий. Информация о каждом отдельном пальце хранится в форме объекта Path (пакет android.graphics), представляющего собой геометрический контур, состоящий из отрезков прямых линий и кривых. События, связанные с касанием пальцами экрана, обрабатываются путем переопределения метода OnTouchEvent класса View (раздел 9.5.2). Этот метод получает объект MotionEvent (пакет android.view), который включает тип касания, происшедшее событие и ID (идентификатор) пальца (то есть указатель), который сгенерировал событие. С помощью ID идентифицируются различные пальцы, а также добавляются сведения в соответствующие объекты Path. С помощью типа события касания экрана определяется, *коснулся* ли пользователь экрана, выполнил ли он *перетаскивание* по поверхности экрана либо *оторвал* палец от экрана.

Сохранение рисунка в галерее устройства

Приложение поддерживает элемент меню Save Image (Сохранить изображение), с помощью которого можно сохранить изображение в галерее устройства — заданном по

умолчанию местоположении, предназначенном для хранения изображений и фотографий. С помощью объекта ContentResolver (пакет android.content) приложение считывает данные с устройства, а также сохраняет их на этом же устройстве. Этот объект используется для получения потока OutputStream (раздел 9.5.2), применяемого для сохранения данных в галерее, а также для сохранения изображений в формате JPEG.

Использование объектов Toast для кратковременного отображения сообщений

Объект Toast (пакет android.widget) отображает сообщение на краткое время, которое затем исчезает с экрана. Эти объекты часто используются для отображения сообщений о незначительных ошибках либо информационных сообщений, которые, например, уведомляют пользователя о выполнении обновления данных. Подобный объект будет использован (раздел 9.5.2) для идентификации успешного сохранения рисунка в галерее.

9.4. Создание графического интерфейса пользователя и файлов ресурсов приложения

В этом разделе вы создадите файлы ресурсов и разметки GUI приложения Doodlz.

9.4.1. Создание проекта

Начните с создания нового проекта Android под названием Doodlz. В диалоговом окне New Android Project (Новый проект Android) укажите следующие значения, затем нажмите кнопку Finish (Готово).

- Build Target (Операционная система): Android 2.3.3.
- Application name (Имя приложения): Doodlz.
- Package name (Название пакета): com.deitel.doodlz.
- Create Activity (Создать действие): Doodlz.
- Min SDK Version (Минимальная версия SDK): 8.

9.4.2. Файл AndroidManifest.xml

В листинге 9.1 приведен код файла AndroidManifest.xml, используемого в приложении. В этом приложении атрибуту android:targetSdkVersion элемента uses-sdk присваивается значение "11" (строка 15), которое соответствует библиотеке Android 3.0 SDK. Если это приложение установлено на устройстве с Android 3.0 либо более старшей версии, к компонентам GUI этого приложения может применяться «голографическая» тема Android 3.0, а элементы меню могут помещаться в правой части меню действий приложения, которое отображается в верхней части экрана планшетов. Присваивание атрибуту android:targetSdkVersion значения 11 не даст эффекта, если приложение установлено на устройстве с более ранней версией Android. Выбор версии SDK 11 рекомендуется для любых приложений, устанавливаемых на планшетах Android. В результате вы

получите тот же эффект, что и в случае использования приложений, разработанных для Android 3.0 или более поздней версии.

Листинг 9.1. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android
   android:versionCode="1" android:versionName="1.0"
   package="com.deitel.doodlz">
   <application android:icon="@drawable/icon"
      android:label="@string/app_name" android:debuggable="true">
         <activity android:label="@string/app_name"
            android:name=".Doodlz"
            android:screenOrientation="portrait">
            <intent-filter>
              <action android:name="android.intent.action.MAIN" />
               <category android:name=
                  "android.intent.category.LAUNCHER"/>
            </intent-filter>
         </activity>
      </application>
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="11" />
 </manifest>
```

9.4.3. Файл strings.xml

В листинге 9.2 определяются ресурсы String, используемые в этом приложении.

Листинг 9.2. Ресурсы Strings, определенные в файле strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">Doodlz</string>
   <string name="button_erase">Erase</string>
   <string name="button_cancel">Cancel</string>
   <string name="button_set_color">Set Color</string>
   <string name="button_set_line_width">Set Line Width</string>
   <string name="label_alpha">Alpha</string>
   <string name="label_red">Red</string>
   <string name="label_green">Green</string>
   <string name="label_blue">Blue</string>
   <string name="menuitem_clear">Clear</string>
   <string name="menuitem_color">Color</string>
   <string name="menuitem_erase">Erase</string>
   <string name="menuitem_line_width">Line Width</string>
   <string name="menuitem_save_image">Save Image</string>
   <string name="message_erase">Erase the drawing?</string>
   <string name="message_error_saving">
      There was an error saving the image</string>
   <string name="message_saved">
      Your painting has been saved to the Gallery</string>
   <string name="title_color_dialog">Choose Color</string>
```

```
  <string name="title_line_width_dialog">Choose Line Width</string>
</resources>
```

9.4.4. Файл main.xml

Заданный по умолчанию файл main.xml был заменен новым файлом. В рассматриваемом случае в проект добавляется единственный компонент разметки, представляющий собой пользовательский подкласс класса View, который называется DoodleView. В листинге 9.3 представлен весь файл main.xml, в который вручную добавлен XML-элемент (строки 2–5). Обратите внимание, что пользовательский класс DoodleView отсутствует в палитре модуля ADT, поэтому его невозможно перетащить в структуру.

Листинг 9.3. XML-разметка приложения Doodlz (main.xml)

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <com.deitel.doodlz.DoodleView "
3     xmlns:android=http://schemas.android.com/apk/res/android
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"/>
```

9.4.5. Файл color_dialog.xml

В листинге 9.4 приведен завершенный код файла color_dialog.xml, определяющего графический интерфейс пользователя для диалогового окна, в котором пользователь может определять компоненты альфа, красный, зеленый и синий для цвета, применяемого в рисовании. Компонент LinearLayout (строки 61–67) имеет белый фон и содержит компонент View (строки 64–66), используемый для отображения текущего цвета рисования, основанного на значениях четырех ползунков SeekBar. Каждый из этих ползунков обеспечивает возможность выбора значений от 0 (заданный по умолчанию минимум) до 255 (определенный максимум). Если пользователь с помощью компонента alphaSeekBar придаст цвету полупрозрачность, благодаря белому фону будет обеспечена точная передача цвета компонентом View. В разрабатываемых в этой главе приложениях используются стандартные ползунки SeekBar, настраиваемые путем назначения атрибуту android:thumb компонента SeekBar ресурса drawable, например изображения.

Листинг 9.4. XML-разметка, используемая при создании диалогового окна Choose Color

```
 1  <?xml version="1.0" encoding="utf-8"?>
 2  <LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
 3  android:id="@+id/colorDialogLinearLayout"
 4  android:layout_width="match_parent" android:minWidth="300dp"
 5  android:layout_height="match_parent" android:orientation="vertical">
 6
 7  <TableLayout android:id="@+id/tableLayout"
 8  android:layout_width="match_parent"
 9  android:layout_height="wrap_content" android:layout_margin="10dp"
10  android:stretchColumns="1">
11     <TableRow android:orientation="horizontal"
12        android:layout_width="match_parent"
```

продолжение ⇨

Листинг 9.4 *(продолжение)*

```
         android:layout_height="wrap_content">
         <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/label_alpha" android:gravity="right"
            android:layout_gravity="center_vertical"/>
         <SeekBar android:id="@+id/alphaSeekBar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:max="255"
            android:paddingLeft="10dp" android:paddingRight="10dp"/>
      </TableRow>
      <TableRow android:orientation="horizontal"
         android:layout_width="match_parent"
         android:layout_height="wrap_content">
         <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/label_red" android:gravity="right"
            android:layout_gravity="center_vertical"/>
         <SeekBar android:id="@+id/redSeekBar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:max="255"
            android:paddingLeft="10dp" android:paddingRight="10dp"/>
      </TableRow>
      <TableRow android:orientation="horizontal"
         android:layout_width="match_parent"
         android:layout_height="wrap_content">
         <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/label_green" android:gravity="right"
            android:layout_gravity="center_vertical"/>
         <SeekBar android:id="@+id/greenSeekBar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:max="255"
            android:paddingLeft="10dp" android:paddingRight="10dp"/>
      </TableRow>
      <TableRow android:orientation="horizontal"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content">
         <TextView android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/label_blue" android:gravity="right"
            android:layout_gravity="center_vertical"/>
         <SeekBar android:id="@+id/blueSeekBar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:max="255"
            android:paddingLeft="10dp" android:paddingRight="10dp"/>
      </TableRow>
   </TableLayout>

   <LinearLayout android:background="@android:color/white"
      android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content" android:layout_margin="10dp">
    <View android:id="@+id/colorView"
       android:layout_width="match_parent"
       android:layout_height="30dp"/>
 </LinearLayout>

 <Button android:id="@+id/setColorButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="@string/button_set_color"/>
</LinearLayout>
```

9.4.6. Файл width_dialog.xml

В листинге 9.5 приведен код завершенного файла width_dialog.xml, определяющего графический интерфейс пользователя для диалогового окна, в котором пользователь может выбрать толщину рисуемой линии. При перемещении ползунка widthSeekBar компонент ImageView (строки 6–8) отображает образец линии с выбранными толщиной и цветом.

Листинг 9.5. XML-разметка для диалогового окна Choose Line Width

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
3     android:id="@+id/widthDialogLinearLayout"
4     android:layout_width="match_parent" android:minWidth="300dp"
5     android:layout_height="match_parent" android:orientation="vertical">
6     <ImageView android:id="@+id/widthImageView"
7        android:layout_width="match_parent" android:layout_height="50dp"
8        android:layout_margin="10dp"/>
9     <SeekBar android:layout_height="wrap_content" android:max="50"
10       android:id="@+id/widthSeekBar" android:layout_width="match_parent"
11       android:layout_margin="20dp" android:paddingLeft="20dp"
12       android:paddingRight="20dp"
13       android:layout_gravity="center_horizontal"/>
14    <Button android:id="@+id/widthDialogDoneButton"
15       android:layout_width="wrap_content"
16       android:layout_height="wrap_content"
17       android:layout_gravity="center_horizontal"
18       android:text="@string/button_set_line_width"/>
19 </LinearLayout>
```

9.5. Создание приложения

Это приложение включает два класса — класс Doodlz (подкласс класса Activity; листинги 9.8–9.17) и класс DoodleView (листинги 9.18–9.26).

9.5.1. Подкласс Doodlz класса Activity

Класс Doodlz (листинги 9.6–9.17) — это основной класс Activity приложения Doodlz. Этот класс поддерживает меню и диалоговые окна приложения, а также обработку событий акселерометра.

Операторы package и import

В разделе 9.3 рассматривались новые ключевые классы и интерфейсы, используемые классом Doodlz. Код этих объектов приведен в листинге 9.6.

Листинг 9.6. Операторы package и import класса Doodlz

```
// Doodlz.java
// Создает компонент View, который изменяет цвет в ответ
// на жесты пользователя.
package com.deitel.doodlz;

import java.util.concurrent.atomic.AtomicBoolean;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;

```

Константы и переменные экземпляра класса

В листинге 9.7 определяются переменные и константы из экземпляра класса Doodlz. Переменная doodleView из экземпляра класса DoodleView (строка 32) представляет область рисунка. Интерфейс sensorManager применяется для отслеживания акселерометра, с помощью которого обнаруживается перемещение устройства. Переменные типа float, объявленные в строках 34–36, применяются для вычисления изменений в ускорении

устройства, которые позволяют зафиксировать *событие сотрясения* (при этом пользователю может быть задан вопрос, действительно ли он хочет удалить рисунок). Константа, определенная в строке 47, позволяет отличать малые перемещения от сотрясений. Эта константа появилась вследствие метода «проб и ошибок», который позволил устранить ошибки идентификации сотрясения на различных устройствах. В строке 37 определяется объект AtomicBoolean (по умолчанию принимает значение false), используемый повсеместно в этом классе и позволяющий определить отображение диалогового окна на экране устройства. С помощью этого метода предотвращается одновременное отображение нескольких окон. В строках 40–44 объявляются константы int для пяти элементов меню приложения. Переменная currentDialog класса Dialog применяется для ссылки на диалоговые окна Choose Color либо Choose Line Width (строка 50). В этих окнах пользователь может изменить цвет и толщину линии соответственно.

Листинг 9.7. Поля класса Doodlz

```
30   public class Doodlz extends Activity
31  {
32     private DoodleView doodleView; // создание View
33     private SensorManager sensorManager; // отслеживание акселерометра
34     private float acceleration;          // ускорение
35     private float currentAcceleration;   // текущее ускорение
36     private float lastAcceleration;      // последнее ускорение
37     private AtomicBoolean dialogIsDisplayed = new AtomicBoolean();
                                                            // ложь
38
39     // создание идентификаторов для каждого элемента меню
40     private static final int COLOR_MENU_ID = Menu.FIRST;
41     private static final int WIDTH_MENU_ID = Menu.FIRST + 1;
42     private static final int ERASE_MENU_ID = Menu.FIRST + 2;
43     private static final int CLEAR_MENU_ID = Menu.FIRST + 3;
44     private static final int SAVE_MENU_ID = Menu.FIRST + 4;
45
46     // значение, используемое для идентификации удара устройства
47     private static final int ACCELERATION_THRESHOLD = 15000;
48
49     // переменная, которая ссылается на диалоговые окна Choose Color
       // либо Choose Line Width
50     private Dialog currentDialog;
51
```

Переопределение методов onCreate и onPause класса Activity

Метод onCreate класса Doodlz (см. листинг 9.8) получает ссылку на класс DoodleView, затем инициализирует переменные экземпляра класса, облегчающие вычисление изменения ускорения и позволяющие установить факт сотрясение корпуса устройства в целях удаления рисунка. Изначально переменным currentAcceleration и lastAcceleration присваивается константа GRAVITY_EARTH из SensorManager, с помощью которой ускорение связывается с земной гравитацией. В SensorManager также содержатся константы для других планет Солнечной системы, Луны, а также ряд других полезных чисел. Подробнее см. developer.android.com/reference/android/hardware/SensorManager.html.

Затем в строке 67 вызывается метод enableAccelerometerListening (см. листинг 9.9), с помощью которого выполняется конфигурирование SensorManager для прослушивания событий акселерометра. Метод onPause класса Doodlz (строки 71–76) вызывает метод disableAccelerometerListening (см. листинг 9.9), позволяющий отменить регистрацию обработчика событий акселерометра при переводе приложения в фоновый режим.

Листинг 9.8. Переопределенные методы onCreate и onPause класса Activity

```
52   // вызывается после загрузки Activity
53   @Override
54   protected void onCreate(Bundle savedInstanceState)
55   {
56      super.onCreate(savedInstanceState);
57      setContentView(R.layout.main); // «раздувание» разметки
58
59      // получение ссылок на DoodleView
60      doodleView = (DoodleView) findViewById(R.id.doodleView);
61
62      // инициализация значений ускорения
63      acceleration = 0.00f;
64      currentAcceleration = SensorManager.GRAVITY_EARTH;
65      lastAcceleration = SensorManager.GRAVITY_EARTH;
66
67      enableAccelerometerListening(); // прослушивания тряски
68   }  // конец метода onCreate
69
70   // если приложение находится в фоновом режиме, остановить
     // прослушивание событий сенсора
71   @Override
72   protected void onPause()
73   {
74      super.onPause();
75      disableAccelerometerListening(); // не прослушивать тряску
76   }  // конец метода onPause
77
```

Методы enableAccelerometerListening и disableAccelerometerListening

Метод enableAccelerometerListening (см. листинг 9.9; строки 79–87) применяется для конфигурирования SensorManager. В строках 82–83 с помощью метода getSystemService класса Activity обеспечивается доступ к системной службе SensorManager, с помощью которой приложение взаимодействует с сенсорами устройства. Затем регистрируются принятые события акселерометра. При этом используется метод registerListener класса SensorManager, принимающий следующие три аргумента:

- объект SensorEventListener, отвечающий на события;
- объект Sensor, представляющий данные о типах данных сенсоров, принимаемых приложением. Для выполнения выборки вызывается метод getDefaultSensor класса SensorManager, который передает константу Sensor-type (константа Sensor.TYPE_ACCELEROMETER в данном приложении);

- частота, с которой события передаются приложению. Чтобы получить события со стандартной частотой, выбирается константа SENSOR_DELAY_NORMAL. Повышение частоты будет способствовать получению более точных данных, но при этом возрастает интенсивность использования ресурсов.

Метод disableAccelerometerListening (см. листинг 9.9; строки 90–101), вызываемый с помощью метода onPause, использует метод unregisterListener класса SensorManager для прекращения прослушивания событий акселерометра. Если вы не хотите знать, какое приложение будет переведено в фоновый режим выполнения, присвойте ссылке sensorManager значение null.

Листинг 9.9. Методы enableAccelerometerListening и disableAccelerometerListening

```
 78 // активизация прослушивания событий акселерометра
 79 private void enableAccelerometerListening()
 80 {
 81    // инициализация SensorManager
 82    sensorManager =
 83       (SensorManager) getSystemService(Context.SENSOR_SERVICE);
 84    sensorManager.registerListener(sensorEventListener,
 85       sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
 86    SensorManager.SENSOR_DELAY_NORMAL);
 87 } // конец метода enableAccelerometerListening
 88
 89 // отключение прослушивания событий акселерометра
 90 private void disableAccelerometerListening()
 91 {
 92    // прекращение прослушивания событий сенсора
 93    if (sensorManager != null)
 94    {
 95       sensorManager.unregisterListener(
 96          sensorEventListener,
 97          sensorManager.getDefaultSensor(
 98             SensorManager.SENSOR_ACCELEROMETER));
 99       sensorManager = null;
100    } // конец блока if
101 } // конец описания метода disableAccelerometerListening
102
```

Анонимный внутренний класс, реализующий интерфейс SensorEventListener, предназначенный для обработки событий акселерометра

В листинге 9.10 переопределяется метод onSensorChanged класса SensorEventListener (строки 108–168), обрабатывающий события акселерометра. В процессе перемещения устройства пользователем этот метод определяет, следует ли рассматривать данное перемещение как встряхивание. Если имеет место встряхивание, в строках 133–165 создается и отображается диалоговое окно AlertDialog, в котором пользователю нужно подтвердить удаление рисунка. Интерфейс SensorEventListener также включает метод

onAccuracyChanged (строки 171–174). Поскольку этот метод не применяется в приложении, эти строки остаются пустыми.

Листинг 9.10. Анонимный внутренний класс, реализующий интерфейс SensorEventListener

```
103 // обработчик событий акселерометра
104 private SensorEventListener sensorEventListener =
105    new SensorEventListener()
106 {
107    // использование акселерометра для определения
       // встряхивания устройства
108    @Override
109    public void onSensorChanged(SensorEvent event)
110    {
111       // предотвращение отображения других диалоговых окон
112       if (!dialogIsVisible.get())
113       {
114          // получение значений x, y и z для SensorEvent
115          float x = event.values[0];
116          float y = event.values[1];
117          float z = event.values[2];
118
119          // сохранение предыдущего значения ускорения
120          lastAcceleration = currentAcceleration;
121
122          // вычисление текущего ускорения
123          currentAcceleration = x * x + y * y + z * z;
124
125          // вычисление изменения ускорения
126          acceleration = currentAcceleration *
127             (currentAcceleration – lastAcceleration);
128
129          // если ускорение превышает определенный уровень
130          if (acceleration > ACCELERATION_THRESHOLD)
131          {
132             // создание нового AlertDialog Builder
133             AlertDialog.Builder builder =
134                new AlertDialog.Builder(Doodlz.this);
135
136             // создание сообщения AlertDialog
137             builder.setMessage(R.string.message_erase);
138             builder.setCancelable(true);
139
140             // добавление кнопки Erase
141             builder.setPositiveButton(R.string.button_erase,
142                new DialogInterface.OnClickListener()
143             {
144                public void onClick(DialogInterface dialog, int id)
145                {
146                   dialogIsVisible.set(false);
```

```
147                   doodleView.clear(); // очистка экрана
148                }  // конец метода onClick
149             }  // конец анонимного внутреннего класса
150         ); // завершение вызова setPositiveButton
151
152         // добавление кнопки Cancel
153         builder.setNegativeButton(R.string.button_cancel,
154            new DialogInterface.OnClickListener()
155         {
156            public void onClick(DialogInterface dialog, int id)
157            {
158               dialogIsVisible.set(false);
159               dialog.cancel(); // скрытие диалогового окна
160            }  // конец метода onClick
161         }  // конец анонимного внутреннего класса
162      ); // завершение вызова setNegativeButton
163
164         dialogIsVisible.set(true); // диалоговое окно,
                                       // отображаемое на экране
165         builder.show(); // отображение диалогового окна
166      }  // конец блока if
167   }  // конец блока if
168 } // конец метода onSensorChanged
169
170 // пустое "тело" метода интерфейса SensorEventListener
171 @Override
172 public void onAccuracyChanged(Sensor sensor, int accuracy)
173 {
174    } // конец метода onAccuracyChanged
175 }; // конец анонимного внутреннего класса
176
```

Пользователь может случайно встряхнуть устройство даже в тех случаях, если диалоговые окна отображаются на экране. Поэтому метод onSensorChanged сначала проверяет, отображается ли диалоговое окно. Для этого вызывается метод get класса dialogIsVisible (строка 110). Это гарантирует отсутствие на экране других диалоговых окон, что особенно важно в случае, если события сенсора происходят в другом потоке выполнения. Без выполнения подобной проверки было бы невозможно отобразить окно подтверждения удаления изображения в случае отображения на экране другого диалогового окна.

Параметр SensorEvent содержит информацию о происшедшем изменении состояния сенсора. Предназначенный для хранения информации о событиях акселерометра массив значений этого параметра включает три элемента, которые представляют ускорение (в м/с^2) в направлениях x (влево/вправо), y (вверх/вниз) и z (вперед/назад). Описание и диаграмму системы координат, используемой API SensorEvent, можно найти на веб-сайте: developer.android.com/reference/android/hardware/SensorEvent.html.

Перейдя по этой ссылке, вы также познакомитесь с толкованиями значений x, y и z SensorEvent для различных сенсоров.

Компоненты ускорения сохраняются в строках 115–117, последнее значение переменной currentAcceleration сохраняется в строке 120. В строке 123 суммируются квадраты

компонентов ускорения *x*, *y* и *z*, и сумма сохраняется в переменной currentAcceleration. На основе значений переменных currentAcceleration и lastAcceleration вычисляется значение (ускорение), которое может сравниваться с константой ACCELERATION_THRESHOLD. Если значение переменной превышает константу, это означает, что пользователь перемещает устройство достаточно быстро для того, чтобы это перемещение рассматривалось как встряхивание. В этом случае переменной shakeDetected присваивается значение true, затем конфигурируется и отображается диалоговое окно AlertDialog, в котором пользователь может подтвердить удаление рисунка в результате встряхивания или отменить отображение диалогового окна. Если переменной shakeDetected присваивается значение true, то во время отображения диалогового окна подтверждения метод onSensorChanged не будет отображать другое диалоговое окно, если пользователь встряхивает устройство снова. Если пользователь подтверждает удаление рисунка, в строке 147 вызывается метод clear класса DoodleView (см. листинг 9.20).

ПРИМЕЧАНИЕ

Обработка событий сенсоров или копирование данных событий должны осуществляться как можно быстрее, поскольку значения в массиве сенсоров обновляются для каждого события сенсора.

Методы onCreateOptionsMenu и onOptionsItemSelected

В листинге 9.11 переопределяется метод onCreateOptionsMenu класса Activity, применяемый для настройки меню Activity. Метод add из этого меню применяется для добавления элементов меню (строки 184–193). Ранее уже упоминалось, что первый аргумент этого метода представляет собой идентификатор группы, который может использоваться для группировки элементов группы. Если какие-либо группы отсутствуют, в этом случае для каждого элемента используется константа NONE класса Menu. Второй аргумент — это уникальный идентификатор элемента (одна из констант, определенных в строках 40–44). Третий аргумент определяет порядок следования элемента меню по отношению к другим элементам меню. Поскольку порядок следования элементов меню не столь важен в этом приложении, использовалась константа NONE класса Menu. Это позволяет Android самому решать, как расположить пункты меню. Последний аргумент — это ресурс String, применяемый для отображения каждого элемента меню.

Листинг 9.11. Переопределенные методы onCreateOptionsMenu и onOptionsItemSelected класса Activity

```
177   // отображает параметры конфигурации в меню
178   @Override
179   public boolean onCreateOptionsMenu(Menu menu)
180   {
181      super.onCreateOptionsMenu(menu); // вызов метода суперкласса
182
183      // добавление параметров в меню
184      menu.add(Menu.NONE, COLOR_MENU_ID, Menu.NONE,
185         R.string.menuitem_color);
186      menu.add(Menu.NONE, WIDTH_MENU_ID, Menu.NONE,
187         R.string.menuitem_line_width);
```

```
188       menu.add(Menu.NONE, ERASE_MENU_ID, Menu.NONE,
189          R.string.menuitem_erase);
190       menu.add(Menu.NONE, CLEAR_MENU_ID, Menu.NONE,
191          R.string.menuitem_clear);
192       menu.add(Menu.NONE, SAVE_MENU_ID, Menu.NONE,
193          R.string.menuitem_save_image);
194
195       return true; // обработано создание параметров меню
196    }  // завершение метода onCreateOptionsMenu
197
198    // обработка выбранных параметров меню
199    @Override
200    public boolean onOptionsItemSelected(MenuItem item)
201    {
202       // оператор switch, использующий MenuItem id
203       switch (item.getItemId())
204       {
205          case COLOR_MENU_ID:
206             showColorDialog(); // диалоговое окно выбора цвета
207             return true; // результат обработки события меню
208          case WIDTH_MENU_ID:
209             showLineWidthDialog(); // диалоговое окно выбора
                                       // толщины линии
210             return true; // результат обработки события меню
211          case ERASE_MENU_ID:
212             doodleView.setDrawingColor(Color.WHITE); // белый цвет линии
213             return true; // результат обработки события меню
214          case CLEAR_MENU_ID:
215             doodleView.clear(); // очистка doodleView
216             return true; // результат обработки события меню
217          case SAVE_MENU_ID:
218             doodleView.saveImage(); // сохранение текущих изображений
219             return true; // результат обработки события меню
220       }  // конец блока switch
221
222       return super.onOptionsItemSelected(item); // вызов метода
                                                    // суперкласса
223    }  // конец метода onOptionsItemSelected
224
```

В строках 199–223 переопределяется метод onOptionItemSelected класса Activity, который вызывается после выбора пользователем элемента меню. С помощью ID аргумента MenuItem (строка 203) выполняются различные действия в зависимости от элемента, выбранного пользователем. Эти действия приведены в следующем списке:

- Color. В строке 206 вызывается метод showColorDialog (см. листинг 9.12), с помощью которого пользователь выбирает новый цвет.
- Width. В строке 209 вызывается метод showLineWidthDialog (см. листинг 9.15), с помощью которого пользователь может выбрать новое значение ширины линии.
- Erase. В строке 212 выбирается белый цвет рисования для doodleView, в результате чего пальцы пользователя превращаются в «ластики».

- Clear. В строке 215 вызывается метод clear класса doodleView, с помощью которого удаляются все окрашенные линии.
- Save. В строке 218 вызывается метод saveImage класса doodleView для сохранения рисунка в виде изображения в галерее изображений устройства.

Метод showColorDialog

Метод showColorDialog (см. листинг 9.12) создает Dialog и формирует графический интерфейс пользователя путем вызова метода setContentView, с помощью которого «раздувается» файл разметки color_dialog.xml (строки 229–230). Также диалоговому окну присваивается заголовок и статус «отменяемого» — пользователь может нажать кнопку Back устройства, чтобы скрыть диалоговое окно без изменения текущего цвета. В строках 235–242 получаются ссылки на четыре ползунка SeekBar диалогового окна, затем в строках 256–248 каждому компоненту OnSeekBarChangeListener класса SeekBar присваивается значение colorSeekBarChanged (см. листинг 9.13). В строках 251–255 получаем текущий цвет рисования из класса doodleView, который затем используется для установки текущего значения компонента SeekBar. Для извлечения значений ARGB из текущего цвета применяются статические методы alpha, red, green и blue класса Color. Метод setProgress класса SeekBar выполняет позиционирование ползунков. В строках 258–260 получаем ссылку на метод setColorButton диалогового окна и в качестве обработчика событий регистрируем setColorButtonListener (листинг 9.14). Путем присваивания значения true методу set класса DialigVisible определяется отображение диалогового окна (строка 262). И наконец, в строке 263 с помощью метода show отображается диалоговое окно. Новый цвет выбирается только в том случае, если пользователь касается кнопки Set Color (Выбрать цвет) в диалоговом окне.

Листинг 9.12. Название листинга

```
225  // отображает диалоговое окно выбора цвета
226  private void showColorDialog()
227  {
228     // создание диалогового окна и «раздувание» его содержимого
229     currentDialog = new Dialog(this);
230     currentDialog.setContentView(R.layout.color_dialog);
231     currentDialog.setTitle(R.string.title_color_dialog);
232     currentDialog.setCancelable(true);
233
234     // получение ползунков SeekBar цвета и настройка
        // их слушателей onChange
235     final SeekBar alphaSeekBar =
236        (SeekBar) currentDialog.findViewById(R.id.alphaSeekBar);
237     final SeekBar redSeekBar =
238        (SeekBar) currentDialog.findViewById(R.id.redSeekBar);
239     final SeekBar greenSeekBar =
240        (SeekBar) currentDialog.findViewById(R.id.greenSeekBar);
241     final SeekBar blueSeekBar =
242        (SeekBar) currentDialog.findViewById(R.id.blueSeekBar);
243
244     // регистрация слушателей событий SeekBar
245     alphaSeekBar.setOnSeekBarChangeListener(colorSeekBarChanged);
```

```
246      redSeekBar.setOnSeekBarChangeListener(colorSeekBarChanged);
247      greenSeekBar.setOnSeekBarChangeListener(colorSeekBarChanged);
248      blueSeekBar.setOnSeekBarChangeListener(colorSeekBarChanged);
249
250      // использование текущего цвета рисунка для выбора значений SeekBar
251      final int color = doodleView.getDrawingColor();
252      alphaSeekBar.setProgress(Color.alpha(color));
253      redSeekBar.setProgress(Color.red(color));
254      greenSeekBar.setProgress(Color.green(color));
255      blueSeekBar.setProgress(Color.blue(color));
256
257      // настройка слушателей кнопки onClickListeneset для класса Color
258      Button setColorButton = (Button) currentDialog.findViewById(
259         R.id.setColorButton);
260      setColorButton.setOnClickListener(setColorButtonListener);
261
262      dialogIsVisible.set(true); // диалоговое окна на экране
263      currentDialog.show();      // отображение диалогового окна
264   }  // конец метода showColorDialog
265
```

Анонимный внутренний класс, реализующий интерфейс OnSeekBarChangeListener для реагирования на события методов alpha, red, green и blue компонентов SeekBar

В листинге 9.13 определяется анонимный внутренний класс, реализующий интерфейс OnSeekBarChangeListener в ответ на события, возникающие в процессе настройки пользователем ползунков SeekBar в диалоговом окне Choose Color. Этот класс регистрируется как обработчик событий компонентов SeekBar (см. листинг 9.12, строки 246–249). В случае изменения положения ползунка SeekBar вызывается метод onProgressChanged (строки 270–290). В диалоговом окне currentDialog выбирается каждый из SeekBar, а также компонент View, применяемый для отображения цвета (строки 275–284). Затем используется метод setBackgroundColor класса View, предназначенный для обновления компонента colorView с помощью компонента Color, который соответствует текущему состоянию компонентов SeekBar (строки 287–289). Статический метод argb класса Color возвращает составленный из значений SeekBar экземпляр Color.

ПРИМЕЧАНИЕ

Метод onProgressChanged вызывается чаще, если пользователь перетаскивает ползунок SeekBar. Исходя из этих соображений, лучше получить ссылки на компонент GUI один раз, а затем хранить их как переменные экземпляра класса вместо того, чтобы получать ссылки при каждом вызове onProgressChanged.

Листинг 9.13. Анонимный внутренний класс, реализующий интерфейс OnSeekbarChangeListener в ответ на события SeekBar в диалоговом окне Choose Color

```
266   // Интерфейс OnSeekBarChangeListener для ползунков
      // SeekBar в диалоговом окне выбора цвета
267   private OnSeekBarChangeListener colorSeekBarChanged =
268      new OnSeekBarChangeListener()
```

продолжение ⇨

Листинг 9.13 *(продолжение)*

```
{
   @Override
   public void onProgressChanged(SeekBar seekBar, int progress,
      boolean fromUser)
   {
   // получение компонентов SeekBar и colorView LinearLayout
   SeekBar alphaSeekBar =
      (SeekBar) currentDialog.findViewById(R.id.alphaSeekBar);
   SeekBar redSeekBar =
      (SeekBar) currentDialog.findViewById(R.id.redSeekBar);
   SeekBar greenSeekBar =
      (SeekBar) currentDialog.findViewById(R.id.greenSeekBar);
   SeekBar blueSeekBar =
      (SeekBar) currentDialog.findViewById(R.id.blueSeekBar);
   View colorView =
      (View) currentDialog.findViewById(R.id.colorView);

   // отображение текущего цвета
   colorView.setBackgroundColor(Color.argb(
      alphaSeekBar.getProgress(), redSeekBar.getProgress(),
      greenSeekBar.getProgress(), blueSeekBar.getProgress()));
   }  // конец метода onProgressChanged

   // требуется указать метод для интерфейса OnSeekBarChangeListener
   @Override
   public void onStartTrackingTouch(SeekBar seekBar)
   {
   }   // конец метода onStartTrackingTouch

   // метод, требуемый для интерфейса OnSeekBarChangeListener
   @Override
   public void onStopTrackingTouch(SeekBar seekBar)
   {
   } // конец метода onStopTrackingTouch
}; // конец colorSeekBarChanged
```

Анонимный внутренний класс, реализующий интерфейс OnClickListener, который используется для настройки нового цвета

В листинге 9.14 определяется анонимный внутренний класс, реализующий интерфейс OnClickListener. Этот интерфейс применяется для выбора нового цвета после щелчка пользователя на кнопке Set Color в диалоговом окне Choose Color. Данный интерфейс регистрируется в качестве обработчика событий класса Button в листинге 9.12 (строка 261). Метод onClick получает ссылки на компоненты SeekBar, затем использует их в строках 322–324 для получения значения каждого ползунка SeekBar и выбора нового цвета. В строке 325 методу set класса isDialigVisible присваивается значение false, означающее, что диалоговое окно не будет отображаться. В строке 326 вызывается метод dismiss класса Dialog, который выполняет закрытие окна и возврат в приложение.

Листинг 9.14. Анонимный внутренний класс, реализующий интерфейс OnClickListener, реагирующий на касание пользователем кнопки Set Color

```
// Интерфейс OnClickListener, используемый для выбора цвета
// после выбора кнопки Set Color в диалоговом окне
private OnClickListener setColorButtonListener = new OnClickListener()
{
   @Override
   public void onClick(View v)
   {
      // получение цвета SeekBar
      SeekBar alphaSeekBar =
         (SeekBar) currentDialog.findViewById(R.id.alphaSeekBar);
      SeekBar redSeekBar =
         (SeekBar) currentDialog.findViewById(R.id.redSeekBar);
      SeekBar greenSeekBar =
         (SeekBar) currentDialog.findViewById(R.id.greenSeekBar);
      SeekBar blueSeekBar =
         (SeekBar) currentDialog.findViewById(R.id.blueSeekBar);

      // выбор цвета линии
      doodleView.setDrawingColor(Color.argb(
         alphaSeekBar.getProgress(), redSeekBar.getProgress(),
         greenSeekBar.getProgress(), blueSeekBar.getProgress()));
      dialogIsVisible.set(false); // диалоговое окно не на экране
      currentDialog.dismiss();    // скрытие диалогового окна
      currentDialog = null;       // диалоговое окно не нужно
   } // конец метода onClick
}; // конец определения интерфейса setColorButtonListener

```

Метод showLineWidthDialog

Метод showLineWidthDialog (см. листинг 9.15) создает диалоговое окно и настраивает соответствующий графический интерфейс пользователя. При этом вызывается метод setContentView, «раздувающий» файл разметки width_dialog.xml (строки 335–336). Также диалоговому окну присваивается заголовок и указывается, что оно является «отменяемым». В строках 341–344 получаются ссылки на ползунки SeekBar в диалоговом окне, в качестве интерфейса OnSeekBarChangeListener выбирается widthSeekBarChanged listener (см. листинг 9.16), которому назначается текущее значение. В строках 347–349 получаем ссылку на объект Button в диалоговом окне, а в качестве OnClickListener выбирается setLineWidthButtonListener (см. листинг 9.17). В строке 351 определяется, что диалоговое окно будет отображаться путем присваивания методу set класса isDialigVisible значения true. И наконец, оператор в строке 352 отображает диалоговое окно. Линии присваивается новое значение толщины только в том случае, если пользователь нажмет кнопку Set Line Width (Выбрать толщину линии) в диалоговом окне.

Листинг 9.15. Метод showLineWidthDialog создает и отображает диалоговое окно, в котором можно изменить толщину линии

```
331   // отображение диалогового окна, в котором выбирается толщина линии
332   private void showLineWidthDialog()
```

продолжение ⇨

Листинг 9.15 *(продолжение)*

```
{
   // создание диалогового окна и «раздувание» его содержимого
   currentDialog = new Dialog(this);
   currentDialog.setContentView(R.layout.width_dialog);
   currentDialog.setTitle(R.string.title_line_width_dialog);
   currentDialog.setCancelable(true);

   // получение widthSeekBar и его конфигурирование
   SeekBar widthSeekBar =
      (SeekBar) currentDialog.findViewById(R.id.widthSeekBar);
   widthSeekBar.setOnSeekBarChangeListener(widthSeekBarChanged);
   widthSeekBar.setProgress(doodleView.getLineWidth());

   // Настройка onClickListener для кнопки Set Line Width
   Button setLineWidthButton =
      (Button) currentDialog.findViewById(R.id.widthDialogDoneButton);
   setLineWidthButton.setOnClickListener(setLineWidthButtonListener);

   dialogIsVisible.set(true); // диалоговое окно отображается
                              // на экране
   currentDialog.show(); // отображение диалогового окна
} // конец метода showLineWidthDialog

```

Анонимный внутренний класс, реализующий OnSeekBarChangeListener в ответ на события widthSeekBar

Листинг 9.16 определяет метод widthSeekBarChanged из OnSeekBarChangeListener, отвечающий на события при изменении пользователем компонентов SeekBar в диалоговом окне Choose Line Width. В строках 359–360 создается объект Bitmap, отображающий образец линии и представляющий толщину выбранной линии. В строке 361 создается объект Canvas, предназначенный для рисования на Bitmap. Метод onProgressChanged (строки 364–381) рисует образец линии, основываясь на текущем выбранном цвете и значении SeekBar. В строках 368–369 получается ссылка на компонент ImageView, в котором отображается линия. Затем в строках 372–375 конфигурируется объект Paint, применяемый для рисования образца линии. Метод setStrokeCap класса Paint (строка 374) определяет видимость концов линии — в рассматриваемом случае они скруглены (определяются с помощью параметра Paint.Cap.ROUND). В строке 378 очищается фон растра (окрашивается в белый цвет) с помощью метода eraseColor класса Bitmap. С помощью холста рисуется образец линии. И наконец, в строке 380 отображается растр с помощью компонента widthImageView путем его передачи методу setImageBitmap компонента ImageView.

Листинг 9.16. Анонимный внутренний класс, реализующий интерфейс OnSeekbarChangeListener в ответ на события SeekBar в диалоговом окне Choose Line Width

```
355   // Интерфейс OnSeekBarChangeListener для компонента
      // SeekBar в диалоговом окне width
356   private OnSeekBarChangeListener widthSeekBarChanged =
```

```
357      new OnSeekBarChangeListener()
358      {
359         Bitmap bitmap = Bitmap.createBitmap( // создание Bitmap
360            400, 100, Bitmap.Config.ARGB_8888);
361         Canvas canvas = new Canvas(bitmap);  // связывание
                                                 // с объектом Canvas
362
363         @Override
364         public void onProgressChanged(SeekBar seekBar, int progress,
365            boolean fromUser)
366         {
367         // получение ImageView
368         ImageView widthImageView = (ImageView)
369            currentDialog.findViewById(R.id.widthImageView);
370
371         // конфигурирование объекта Paint для текущего значения SeekBar
372         Paint p = new Paint();
373         p.setColor(doodleView.getDrawingColor());
374         p.setStrokeCap(Paint.Cap.ROUND);
375         p.setStrokeWidth(progress);
376
377         // очистка растра и перерисовывание линии
378         bitmap.eraseColor(Color.WHITE);
379         canvas.drawLine(30, 50, 370, 50, p);
380         widthImageView.setImageBitmap(bitmap);
381      }  // конец определения метода onProgressChanged
382
383      // метод, требуемый для интерфейса OnSeekBarChangeListener
384      @Override
385      public void onStartTrackingTouch(SeekBar seekBar)
386      {
387      } // завершение определения метода onStartTrackingTouch
388
389      // метод, требуемый для интерфейса OnSeekBarChangeListener
390      @Override
391      public void onStopTrackingTouch(SeekBar seekBar)
392      {
393      } // конец определения метода onStopTrackingTouch
394      }; // конец определения метода widthSeekBarChanged
395
```

Анонимный внутренний класс, реализующий интерфейс OnClickListener в ответ на события кнопки Set Line Width

В листинге 9.17 определяется анонимный внутренний класс, реализующий интерфейс OnClickListener. С помощью этого интерфейса можно выбрать толщину новой цветной линии после щелчка пользователя на кнопке Set Line Width (Настроить толщину линии) в диалоговом окне Choose Line Width (Выбор толщины линии). Этот интерфейс регистрируется в качестве обработчика событий класса Button в листинге 9.15 (строка 349). Метод onClick получает ссылку на компонент SeekBar в диалоговом окне, а затем использует ее для настройки ширины новой линии на основании значения SeekBar. В строке 409

блокируется отображение диалогового окна путем вызова метода set класса isDialigVisible с параметром false. В строке 410 вызывается метод dismiss класса Dialog, который закрывает диалоговое окно и выполняет возврат в окно приложения.

Листинг 9.17. Анонимный внутренний класс, реализующий интерфейс OnClickListener, с помощью которого обрабатываются события кнопки Set Line Width

```
396  // Интерфейс OnClickListener, выполняющий настройку ширины линии
     // после щелчка на кнопке Set Line Width
397  private OnClickListener setLineWidthButtonListener =
398     new OnClickListener()
399  {
400     @Override
401     public void onClick(View v)
402     {
403        // получение цвета с помощью SeekBar
404        SeekBar widthSeekBar =
405           (SeekBar) currentDialog.findViewById(R.id.widthSeekBar);
406
407        // настройка цвета линии
408        doodleView.setLineWidth(widthSeekBar.getProgress());
409        dialogIsVisible.set(false); // диалоговое окно не на экране
410        currentDialog.dismiss();    // скрытие диалогового окна
411        currentDialog = null;       // диалоговое окно не нужно
412      } // конец описания метода onClick
413   };// конец описания интерфейса setColorButtonListener
414 } // конец описания класса Doodlz
```

9.5.2. Подкласс DoodleView класса View

Класс DoodleView (см. листинги 9.18–9.26) обрабатывает жесты пользователей и рисует соответствующие линии.

Класс DoodleView для приложения Doodlz: основной «раскрашиваемый» экран

В листинге 9.18 приведены операторы package и import, а также поля для класса DoodleView из приложения Doodlz. Новые классы и интерфейсы были рассмотрены в разделе 9.3 и упоминаются в настоящем разделе.

Листинг 9.18. Операторы package и import класса DoodleView

```
// DoodleView.java
// Основное представление для приложения Doodlz.
package com.deitel.doodlz;

import java.io.IOException;
import java.io.OutputStream;
import java.util.HashMap;

import android.content.ContentValues;
```

```
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.Point;
import android.net.Uri;
import android.provider.MediaStore.Images;
import android.util.AttributeSet;
import android.view.Gravity;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Toast;

```

Поля, конструктор и метод onSizeChanged класса DoodleView

Поля класса DoodleView (см. листинг 9.19, строки 29–36) применяются для управления данными набора линий, используемых пользователем, а также для рисования этих линий. Конструктор (строки 39–54) инициализирует поля класса. В строке 43 создается объект paintScreen класса Paint, который будет использоваться для отображения рисунка пользователя на экране. В строке 46 создается объект paintLine класса Paint, который определяет настройки линии(й), которую рисует пользователь в данный момент времени. В строках 47–51 определяются настройки объекта paintLine. Методу setAntiAlias класса Paint передается значение true, с помощью которого выполняется *сглаживание* концов линий. Затем методу style класса Paint передается параметр Paint.Style.STROKE (при этом используется метод setStyle класса Paint). Методу style могут передаваться параметры STROKE, FILL или FILL_AND_STROKE для линии, залитой фигуры без границ и залитой фигуры с границами соответственно. По умолчанию используется параметр Paint.Style.FILL. С помощью метода setStrokeWidth класса Paint настраивается толщина линии. По умолчанию в приложении выбирается толщина линии, равная пяти пикселям. Мы также используем метод setStrokeCap класса Paint для округления концов линии с помощью параметра Paint.Cap.ROUND. В строке 52 создается метод pathMap, который отображает идентификатор каждого пальца (известен как указатель) на соответствующий объект Path для линий, которые уже нарисованы. В строке 53 создается метод previousPointMap, который следит за положением каждого пальца — по мере перемещения пальца рисуется линия от текущей до предыдущей точки, выбранных с помощью пальца.

Листинг 9.19. Поля, конструктор и переопределенный метод onSizeChanged класса DoodleView

```
25   // Главный раскрашиваемый экран
26   public class DoodleView extends View
27   {
28      // определяется, переместил ли пользователь палец на расстояние,
        // достаточное для повторного рисования
29      private static final float TOUCH_TOLERANCE = 10;
30
31      private Bitmap bitmap; // область для отображения или сохранения
```

продолжение ☞

Листинг 9.19 *(продолжение)*

```
32      private Canvas bitmapCanvas; // рисование на растре
33      private Paint paintScreen;   // рисование растра на экране
34      private Paint paintLine;     // рисование линий на растре
35      private HashMap<Integer, Path> pathMap; // рисование текущего Paths
36      private HashMap<Integer, Point> previousPointMap; // текущие Points
37
38      // Конструктор DoodleView инициализирует DoodleView
39      public DoodleView(Context context, AttributeSet attrs)
40      {
41         super(context, attrs); // передача содержимого конструктору View
42
43         paintScreen = new Paint(); // применяется для отображения
                                      // растра на экране
44
45         // настройка начальных настроек отображения
           // для нарисованной линии
46         paintLine = new Paint();
47         paintLine.setAntiAlias(true); // сглаживание краев
                                         // нарисованной линии
48         paintLine.setColor(Color.BLACK); // по умолчанию выбран черный
49         paintLine.setStyle(Paint.Style.STROKE); // сплошная линия
50         paintLine.setStrokeWidth(5); // настраивается заданная
                                        // по умолчанию ширина линии
51         paintLine.setStrokeCap(Paint.Cap.ROUND); // скругленные концы
                                                    // линии
52         pathMap = new HashMap<Integer, Path>();
53         previousPointMap = new HashMap<Integer, Point>();
54     }   // завершение описания конструктора DoodleView
55
56     // Метод onSizeChanged создает BitMap и Canvas
       // после отображения приложения
57     @Override
58     public void onSizeChanged(int w, int h, int oldW, int oldH)
59     {
60        bitmap = Bitmap.createBitmap(getWidth(), getHeight(),
61           Bitmap.Config.ARGB_8888);
62        bitmapCanvas = new Canvas(bitmap);
63        bitmap.eraseColor(Color.WHITE); // заливка BitMap белым цветом
64    }   // конец описания метода onSizeChanged
65
```

Размер компонента DoodleView не определен до тех пор, пока этот компонент не будет «раздут» и добавлен в иерархию классов View для класса Activity приложения Doodlz. В результате невозможно определить размер объекта Bitmap, с помощью которого выполняется рисование, посредством метода onCreate. Чтобы разрешить эту проблему, в строках 58–64 переопределяется метод onSizeChanged класса View, который вызывается в случае изменения размеров компонента DoodleView. Это имеет место, например, при добавлении этого компонента в иерархию классов View класса Activity или при вращении устройства пользователем. В этом приложении метод onSizeChanged вызывается только

в том случае, если класс DoodleView добавлен в иерархию классов View класса Activity приложения Doodlz, поскольку приложение всегда выводит изображение в портретном режиме (см. листинг 9.1). Статический метод createBitmap класса Bitmap создает растр, имеющий указанные значения ширины и высоты. В данном случае в качестве размеров объекта Bitmap применяются значения ширины и высоты класса DoodleView. Последний аргумент метода createBitmap — это encoding (кодировка) класса Bitmap, который определяет, каким образом хранится каждый пиксель в растре. Константа Bitmap.Config.ARGB_8888 указывает, каким образом хранится цвет каждого пикселя в четырех байтах (по одному байту для альфа, красного, зеленого и синего значений цвета пикселя). Затем будет создан новый объект Canvas, использующийся для рисования фигур непосредственно на Bitmap. И наконец, воспользуемся методом eraseColor класса Bitmap для заливки растра белыми пикселями (по умолчанию фон растра является черным).

Методы clear, setDrawingColor, getDrawingColor, setLineWidth и getLineWidth класса DoodleView

В листинге 9.20 определяются методы clear (строки 67–73), setDrawingColor (строки 76–79), getDrawingColor (строки 82–85), setLineWidth (строки 88–91) и getLineWidth (строки 94–97), которые вызываются из класса Activity приложения Doodlz. Метод clear очищает pathMap и previousPointMap, очищает растр путем выбора белого цвета для всех его пикселей, а затем вызывается унаследованный invalidate класса View, с помощью которого определяется, нужно ли перерисовывать класс View. Затем система автоматически определяет момент вызова метода onDraw класса View. Метод setDrawingColor изменяет текущий цвет рисунка путем выбора цвета для объекта paintLine класса Paint. Метод setColor класса Paint получает значение переменной типа int, которая представляет новый цвет в формате ARGB. Метод getDrawingColor возвращает текущий цвет, использующийся в диалоговом окне Choose Color. Метод setLineWidth применяется для настройки толщины штриха объекта paintLine, равной определенному количеству пикселей. Метод getLineWidth возвращает текущую толщину штриха, которая используется в диалоговом окне Choose Line Width (Выбор толщины линии).

Листинг 9.20. Методы clear, setDrawingColor, getDrawingColor, setLineWidth и getLineWidth класса DoodleView

```
66   // очистка рисунка
67   public void clear()
68   {
69      pathMap.clear(); // удаление всех контуров
70      previousPointMap.clear(); // удаление всех предыдущих точек
71      bitmap.eraseColor(Color.WHITE); // очистка растра
72      invalidate(); // обновление экрана
73   }  // завершение описания метода clear
74
75   // настройка цвета нарисованной линии
76   public void setDrawingColor(int color)
77   {
78      paintLine.setColor(color);
79   }  // завершение описания метода setDrawingColor
80
```

продолжение ⇗

Листинг 9.20 *(продолжение)*

```
// возврат цвета нарисованной линии
public int getDrawingColor()
{
   return paintLine.getColor();
}  // завершение описания метода getDrawingColor

// выбор толщины нарисованной линии
public void setLineWidth(int width)
{
   paintLine.setStrokeWidth(width);
}  // завершение описания метода setLineWidth

// возврат толщины нарисованной линии
public int getLineWidth()
{
   return (int) paintLine.getStrokeWidth();
}  // завершение описания метода getLineWidth

```

Переопределение метода OnDraw класса View

Если объект View нужно перерисовать, вызывается его метод onDraw. В листинге 9.21 переопределяется метод onDraw, используемый для отображения растра (объект Bitmap, который содержит рисунок), с помощью компонента DoodleView. При этом вызывается метод drawBitmap с аргументом Canvas. Первый аргумент — это растр, на котором выполняется рисование. Следующие два аргумента — это координаты *x-y* верхнего левого угла растра, который должен быть помещен в области компонента View. Последний аргумент — это объект Paint, определяющий характеристики рисунка. Затем в строках 107–108 выполняется цикл по каждому целочисленному ключу в HashMap из pathMap. Для каждого ключа передается соответствующий аргумент Path методу drawPath класса Canvas. При этом рисуются контуры на экране с помощью объекта paintLine, который определяет толщину и цвет линии.

Листинг 9.21. Переопределенный метод onDraw класса DoodleView

```
 99 // вызывается при каждом рисовании в данном представлении
100 @Override
101 protected void onDraw(Canvas canvas)
102 {
103    // рисование фонового экрана
104    canvas.drawBitmap(bitmap, 0, 0, paintScreen);
105
106    // для каждого только что нарисованного контура
107    for (Integer key : pathMap.keySet())
108       canvas.drawPath(pathMap.get(key), paintLine); // рисование линии
109    }  // завершение описания метода onDraw
110
```

Переопределение метода onTouchEvent класса View

Метод OnTouchEvent (см. листинг 9.22) вызывается в случае, если класс View принимает событие касания. В Android поддерживается мультитач-технология, то есть распознаются жесты, заключающиеся в одновременном касании экрана несколькими пальцами. Пользователь может коснуться экрана несколькими пальцами либо убрать пальцы с экрана в любой момент времени. Для реализации этой технологии каждому пальцу, известного под названием *указатель*, назначается уникальный идентификатор, который позволяет идентифицировать связанное с этим пальцем событие касания экрана. Идентификатор будет использоваться для поиска соответствующих объектов Path, представляющих каждую нарисованную линию. Эти объекты хранятся в библиотеке pathMap.

Листинг 9.22. DoodleView переопределяется методом onTouchEvent

```
111   // обработка событий касания экрана
112   @Override
113   public boolean onTouchEvent(MotionEvent event)
114   {
115      // получение типа события и идентификатора указателя,
         // который вызвал событие
116      int action = event.getActionMasked();     // тип события
117      int actionIndex = event.getActionIndex(); // указатель (палец)
118
119      // определение типа действия для данного MotionEvent
120      // представляет, затем вызывает соответствующий метод обработки
121      if (action == MotionEvent.ACTION_DOWN ||
122         action == MotionEvent.ACTION_POINTER_DOWN)
123         {
124            touchStarted(event.getX(actionIndex),
                  event.getY(actionIndex),
125               event.getPointerId(actionIndex));
126         }  // завершение описания блока if
127      else if (action == MotionEvent.ACTION_UP ||
128         action == MotionEvent.ACTION_POINTER_UP)
129         {
130            touchEnded(event.getPointerId(actionIndex));
131         }  // конец описания блока else if
132         else
133         {
134            touchMoved(event);
135         }  // конец описания блока else
136
137         invalidate(); // перерисовывание
138         return true;  // использование события касания
139     }   // конец описания метода onTouchEvent
140
```

Метод getActionMasked класса MotionEvent (строка 116) возвращает значение типа int, представляющее тип MotionEvent. Этот тип можно использовать вместе с константами класса MotionEvent для определения порядка обработки каждого события. Метод getActionIndex класса MotionEvent возвращает целочисленный индекс, соответствующий

пальцу, вызвавшему событие. Этот индекс является не уникальным идентификатором пальца, а всего лишь средством передачи информации о пальце, находящейся в объекте MotionEvent. Чтобы получить уникальный идентификатор пальца, который остается постоянным для всех объектов MotionEvent до тех пор, пока пользователь не уберет палец с экрана, используется метод getPointerID класса MotionEvent (строки 125 и 130), передающий индекс пальца в качестве аргумента.

Генерация действия MotionEvent.ACTION_DOWN либо MotionEvent.ACTION_POINTER_DOWN (строки 121–122) означает, что пользователь коснулся экрана новым пальцем. Первый палец, коснувшийся экрана, генерирует событие MotionEvent.ACTION_DOWN, а другие пальцы — события MotionEvent. ACTION_POINTER_DOWN. В этих случаях вызывается метод touchStarted (см. листинг 9.23), с помощью которого сохраняются начальные координаты прикосновения. Если генерируется событие MotionEvent.ACTION_UP либо MotionEvent. ACTION_POINTER_UP, это означает, что пользователь убрал палец от экрана. В этом случае вызывается метод touchEnded (см. листинг 9.25), который рисует завершенный контур на растре, в результате чего появляется постоянная запись в классе Path. В случае остальных событий касания вызывается метод touchMoved (см. листинг 9.24), с помощью которого рисуются линии. После обработке события в строке 137 вызывается унаследованный метод invalidate класса View, перерисовывающий экран, а в строке 138 возвращается значение true, свидетельствующее о том, что событие было обработано.

Метод touchStarted класса DoodleView

Метод утилиты touchStarted (см. листинг 9.23) вызывается в том случае, если палец первый раз касается экрана. Координаты и идентификатор точки касания поддерживаются в виде аргументов. Если для данного идентификатора уже существует объект Path (строка 148), вызывается метод reset класса Path для удаления существующих точек, в результате чего появляется возможность повторно использовать класс Path для создания нового штриха. В противном случае создается новый объект Path, добавляется в pathMap, а затем новый объект Point добавляется в previousPointMap. В строках 163–165 вызывается метод moveTo класса Path, применяемый для установки начальных координат объекта Path и определения новых значений *x* и *y* для объекта Point.

Листинг 9.23. Метод touchStarted класса DoodleView

```
141  // вызывается после касания пользователем экрана
142  private void touchStarted(float x, float y, int lineID)
143  {
144     Path path; // применяется для хранения контура для
                   // идентификатора данного прикосновения
145     Point point; // применяется для хранения последней точки контура
146
147     // если уже есть контур для идентификатора lineID
148     if (pathMap.containsKey(lineID))
149     {
150        path = pathMap.get(lineID); // получение контура
151        path.reset(); // переустановка контура из-за нового
                         // прикосновения
152        point = previousPointMap.get(lineID); // последняя точка контура
153     }  // конец блока f
```

```
    else
    {
        path = new Path(); // создание нового контура
        pathMap.put(lineID, path); // добавление контура в карту
        point = new Point(); // создание новой точки
        previousPointMap.put(lineID, point);    // добавление точки
                                                // на карту
    }  // конец блока else

    // перемещение координат прикосновения
    path.moveTo(x, y);
    point.x = (int) x;
    point.y = (int) y;
}  // конец описания метода touchStarted
```

Метод touchMoved класса DoodleView

Метод утилиты touchMoved (см. листинг 9.24) вызывается в том случае, если пользователь перемещает один или несколько пальцев вдоль экрана. Объект MotionEvent, передаваемый из метода onTouchEvent, содержит сведения о жестах, если несколько движений на экране происходят одновременно. Метод getPointerCount класса MotionEvent (строка 172) возвращает количество прикосновений, описанных с помощью MotionEvent. Для каждого из прикосновений в pointerID хранится идентификатор пальца (строка 175), а индекс соответствующего пальца для данного MotionEvent (строка 176) хранится в pointerIndex. Затем проверяется наличие соответствующего контура (Path) в pathMap HashMap (строка 179). Если контур обнаружен, с помощью методов getX и getY класса MotionEvent получаем последние координаты для данного события перетаскивания с указанным pointerIndex. Получаем соответствующий контур и последнюю точку (Point) для pointerID из каждой перспективы HashMap, затем вычисляем разницу между последней и текущей точкой. Path обновляется только в том случае, если палец переместился на расстояние, которое превышает значение, определенное константой TOUCH_TOLERANCE. Эти действия направлены на устранение проблем, вызываемых излишней чувствительностью многих устройств, приводящей к генерированию событий MotionEvents даже в случае ничтожно малых перемещений пальца при удержании пальца пользователя в одной точке экрана. Если пользователь смещает палец дальше, чем задано константой TOUCH_TOLERANCE, то метод quadTo класса Path (строки 198–199) рисует между предыдущей и новой точками геометрическую кривую (квадратичную кривую Безье), после чего обновляется последняя точка касания для данного пальца.

Листинг 9.24. Метод touchMoved класса DoodleView

```
168   // вызывается при выполнении перетаскивания в области экрана
169   private void touchMoved(MotionEvent event)
170   {
171   // для каждого из указателей в данном MotionEvent
172   for (int i = 0; i < event.getPointerCount(); i++)
173   {
174      // получение идентификатора и индекса указателя
175      int pointerID = event.getPointerId(i);
176      int pointerIndex = event.findPointerIndex(pointerID);
```

продолжение ⇗

Листинг 9.24 *(продолжение)*

```

      // если имеется контур, связанный с указателем
      if (pathMap.containsKey(pointerID))
      {
         // получение новых координат указателя
         float newX = event.getX(pointerIndex);
         float newY = event.getY(pointerIndex);

         // получение контура и предыдущей точки, связанных
         // с этим указателем
         Path path = pathMap.get(pointerID);
         Point point = previousPointMap.get(pointerID);

         // вычисление перемещения от точки последнего обновления
         float deltaX = Math.abs(newX – point.x);
         float deltaY = Math.abs(newY – point.y);

         // если расстояние достаточно велико
         if (deltaX >= TOUCH_TOLERANCE || deltaY >= TOUCH_TOLERANCE)
         {
            // перемещение контура в новое место
            path.quadTo(point.x, point.y, (newX + point.x) / 2,
               (newY + point.y) / 2);

            // хранение новых координат
            point.x = (int) newX;
            point.y = (int) newY;
         }  // конец определения блока if
      }  // конец блока if
   }  // конец блока for
} // конец определения метода touchMoved

```

Метод touchEnded класса DoodleView

Метод утилиты touchEnded (см. листинг 9.25) вызывается после того, как пользователь оторвет палец от экрана. Этот метод получает идентификатор соответствующего пальца (lineID) в качестве аргумента. В строке 212 получаем соответствующий контур. В строке 213 вызывается метод drawPath класса bitmapCanvas. Этот метод рисует контур на объекте Bitmap, называемого *растром*, перед вызовом метода reset класса Path, применяемого для очистки контура. Переустановка контура не приводит к удалению соответствующей окрашенной линии с экрана, поскольку эти линии уже нарисованы на растре, отображаемом на экране. Линии, которые только что были нарисованы пользователем, отображаются в верхней части этого растра.

Листинг 9.25. Метод touchEnded класса DoodleView

```
209 // вызывается после завершения пользователем прикосновения
210 private void touchEnded(int lineID)
211 {
```

```
      Path path = pathMap.get(lineID); // получение соответствующего
                                       // контура
      bitmapCanvas.drawPath(path, paintLine); // рисует bitmapCanvas
      path.reset(); // переустановка контура
   } // конец описания метода touchEnded

```

Метод saveImage

Метод saveImage (см. листинг 9.26) сохраняет текущий рисунок в файле, находящемся в галерее устройства.

ПРИМЕЧАНИЕ

Вполне возможно, что изображение не появится сразу в галерее. Например, Android сканирует хранилище в поисках новых медиаэлементов, таких как изображения, видеоролики и музыка, после первого включения устройства. Некоторые устройства выполняют сканирование новых медиаресурсов в фоновом режиме. В среде AVD можно запустить приложение AVD Dev Tools и выбрать параметр Media Scanner (Сканер медиаресурсов), после чего новое изображение появится в галерее.

Листинг 9.26. Метод saveImage класса DoodleView

```
   // сохранение текущего изображения в галерее
   public void saveImage()
   {
      // воспользуйтесь "Doodlz" с показаниями времени в качестве
      // имени файла изображения
      String fileName = "Doodlz" + System.currentTimeMillis();

      // создание ContentValues и настройка даты нового изображения
      ContentValues values = new ContentValues();
      values.put(Images.Media.TITLE, fileName);
      values.put(Images.Media.DATE_ADDED, System.currentTimeMillis());
      values.put(Images.Media.MIME_TYPE, "image/jpg");

      // получение Uri местоположения, используемого для хранения файлов
      Uri uri = getContext().getContentResolver().insert(
         Images.Media.EXTERNAL_CONTENT_URI, values);

      try
      {
         // получение OutputStream для uri
         OutputStream outStream =
            getContext().getContentResolver().openOutputStream(uri);

         // копирование растра в OutputStream
         bitmap.compress(Bitmap.CompressFormat.JPEG, 100, outStream);

         // очистка и закрытие потока OutputStream
         outStream.flush();  // очистка буфера
         outStream.close(); // закрытие потока
```

продолжение ⇗

Листинг 9.26 *(продолжение)*

```
245
246         // отображение сообщения о сохранении изображения
247         Toast message = Toast.makeText(getContext(),
248            R.string.message_saved, Toast.LENGTH_SHORT);
249         message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
250            message.getYOffset() / 2);
251         message.show(); // отображение сообщения Toast
252     }   // конец определения блока try
253     catch (IOException ex)
254     {
255        // отображение сообщения о сохранении изображения
256        Toast message = Toast.makeText(getContext(),
257           R.string.message_error_saving, Toast.LENGTH_SHORT);
258        message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
259           message.getYOffset() / 2);
260        message.show();
                            // отображение сообщения Toast
261    }  // конец блока catch
262  } // конец описания метода saveImage
263 } // конец описания класса DoodleView
```

В качестве названия файла изображения используется слово "Doodlz", за которым следует текущее время. В строке 224 создается новый объект ContentValues, который будет использовать ContentResolver для определения заголовка изображения (то есть названия файла), даты создания изображения и типа MIME изображения (в рассматриваемом примере — "image/jpg"). Дополнительные сведения о типах MIME можно найти на веб-сайте: www.w3schools.com/media/media_mimeref.asp.

Метод ContentValues добавляет пару «ключ-значение» к объекту ContentValues. Ключ Images.Media.TITLE (строка 225) применяется для определения fileName в качестве названия файла изображения. Ключ Images.Media.DATE_ADDED (строка 226) применяется для определения времени сохранения файла в памяти устройства. Ключ Images.Media.MIME_TYPE (строка 227) используется для определения в качестве типа файла MIME изображения в формате JPEG.

В строках 230–231 обеспечивается доступ к объекту ContentResolver приложения, затем вызывается его метод insert для получения ссылки Uri, определяющей место сохранения изображения. С помощью константы Images.Media.EXTERNAL_CONTENT_URI определяется, что изображение должно быть сохранено во внешней памяти устройства (обычно на карте памяти SD). Мы передаем ContentValues в качестве второго аргумента для создания файла с выбранным именем файла, датой создания и типом MIME. Как только файл будет создан, сохраним экранный снимок в месте, определенном возвращенной ссылкой Uri. Для этого получаем поток OutputStream, который обеспечивает сохранение снимка в месте, определенном ссылкой Uri (строки 236–237). Затем вызывается метод compress класса Bitmap, который получает константу, представляющую формат сжатия (Bitmap.CompressFormat.JPEG). Эта константа представляет собой качество изображения (значение 100 соответствует изображению лучшего качества). С помощью OutputStream определяется место, используемое для записи байтов изображения. Затем в строках 243–244 выполняется очистка и закрытие OutputStream соответственно.

Если файл сохранен успешно, воспользуемся объектом Toast для сообщения об этом (строки 247–251). Если же при сохранении файла возникла ошибка, объект Toast используется для отображения сообщения об ошибке (строки 256–260). Метод makeText класса Toast получает в качестве аргументов объект Context, текст сообщения и продолжительность отображения этого сообщения. Метод setGravity класса Toast задает место отображения сообщений Toast. Константа Gravity.CENTER указывает, что сообщение Toast должно быть отцентрировано относительно координат, заданных с помощью второго и третьего аргументов. Сообщения Toast отображаются с помощью метода show класса Toast.

9.6. Резюме

При создании приложения из этой главы вы научились превращать экран устройства в виртуальный холст. Была выбрана версия *«11»* библиотеки SDK, благодаря которой приложение, разработанное в версии, предшествующей Android 3.0, может использовать компоненты «голографического» интерфейса пользователя Android 3.0. Также обеспечивается встраивание меню приложения в панель действий Android 3.0 при выполнении приложения на устройстве Android 3.0.

Была выполнена обработка событий сенсора, генерируемых акселерометром устройства, путем регистрации SensorEventListener с помощью системной службы SensorManager. В объектах класса Dialog отображались диалоговые окна с помощью сложных графических интерфейсов пользователя. Для блокирования отображения обработчиком событий сенсора другого диалогового окна в случае отображения текущего диалогового окна использовались безопасные для потока переменные AtomicBoolean.

Вы научились создавать пользовательские цвета ARGB, включающие компоненты альфа, красный, зеленый и синий, а также извлекать отдельные компоненты из существующего цвета. При рисовании линий на растрах использовались связанные объекты Canvas, затем эти растры отображались на экране. Созданные растры сохранялись в качестве изображений в галерее изображений устройства.

В процессе перемещения одного или более пальцев по экрану информация для каждого пальца сохраняется в виде объекта Path. Обработка событий касания экрана осуществляется переопределением метода onTouchEvent класса View и использованием его параметра MotionEvent для получения типа происшедшего события касания, а также идентификатора пальца, который сгенерировал событие.

Вы научились сохранять изображения в галерее устройства с помощью потока OutputStream из ContentResolver. И наконец, использовали объекты Toast для отображения сообщений, исчезающих через короткий промежуток времени.

В главе 10 мы создадим приложение Address Book, обеспечивающее быстрый и простой доступ к сохраненным на устройстве сведениям о контактах. С помощью этого приложения можно удалять и добавлять контакты, а также изменять существующие контакты. Пользователь может выполнять прокрутку списка контактов, отсортированного в алфавитном порядке, добавлять контакты, а также просматривать подробные сведения о выбранном контакте. После касания имени контакта появляется экран, на котором отображаются подробные сведения об адресате.

Приложение Address Book 10

Компоненты ListActivity, AdapterViews, адаптеры, несколько деятельностей, SQLite, стили GUI, ресурсы меню и MenuInflater

В этой главе...

- Расширение класса ListActivity для создания деятельности, по умолчанию состоящей из компонентов ListView.
- Создание нескольких подклассов Activity, представляющих задачи приложения, и их запуск с помощью явно определенных Intents.
- Создание и открытие баз данных SQLite с помощью объекта SQLiteOpenHelper, вставка, удаление и выборка данных в базах данных SQLite с помощью объекта SQLiteDatabase.
- Использование класса SimpleCursorAdapter для связывания результатов запроса базы данных с элементами списка ListView.
- Манипулирование результатами запроса базы данных с помощью класса Cursor.
- Использование многопоточности для выполнения операций с базой данных за пределами потока GUI и поддержки «отзывчивости» приложения.

- Определение стилей, содержащих общие атрибуты и значения GUI, применение стилей для нескольких компонентов GUI.
- Создание ресурсов меню в формате XML, «раздувание» с помощью MenuInflater.

10.1. Введение

Приложение Address Book (рис. 10.1) обеспечивает удобный доступ к сохраненной информации контакта. На главном экране можно выполнить прокрутку отсортированного в алфавитном порядке списка. Чтобы просмотреть подробные сведения о контакте, нажмите его имя. Если нажать кнопку меню устройства во время просмотра сведений, появится меню, содержащее параметры Edit Contact (Изменить контакт) или Delete Contact (Удалить контакт), показанное на рис. 10.2. Если выбрать параметр изменения контакта, приложение запустит объект Activity, отображающий информацию, находящуюся в компонентах EditTexts (см. рис. 10.2). Если пользователь выберет параметр удаления контакта, появится диалоговое окно, в котором отображается запрос подтверждения операции удаления (рис. 10.3). Если во время просмотра списка контактов нажать кнопку меню устройства, появится меню, включающее параметр Add Contact (Добавить контакт). После выбора этого параметра запускается объект Activity, выполняющий добавление нового контакта (рис. 10.4). После касания кнопки Save Contact (Сохранить контакт) добавляется в список новый контакт, а пользователь возвращается к главному экрану контакта.

10.2. Тестирование приложения Address Book

Открытие и выполнение приложения

Откройте среду Eclipse и импортируйте проект Address Book app. Выполните следующие действия:

1. Выполните команды File ▸ Import... (Файл ▸ Импорт...) для открытия диалогового окна Import (Импорт).
2. В диалоговом окне Import раскройте узел General (Общий) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область). Затем щелкните на кнопке Next > (Далее >).
3. Справа от текстового поля Select root directory (Выбрать корневой каталог) щелкните на кнопке Browse... (Просмотр...). Затем найдите и выделите папку AddressBook.
4. Щелкните на кнопке Finish (Готово) для выполнения импорта проекта.

В среде Eclipse щелкните правой кнопкой мыши на проекте приложения, находящемся в окне Package Explorer, затем в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android).

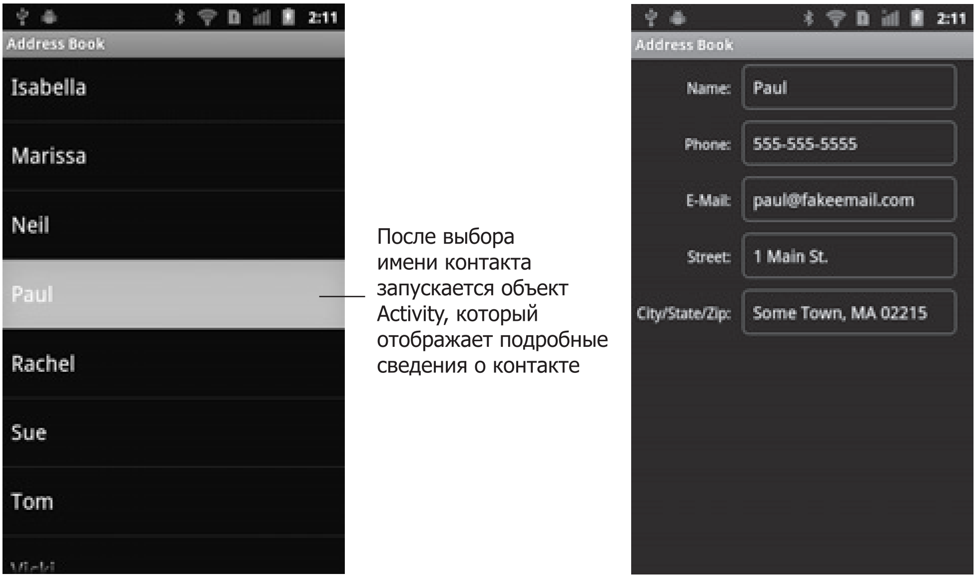


Рис. 10.1. Список с выбранным контактом, для которого отображаются подробные сведения

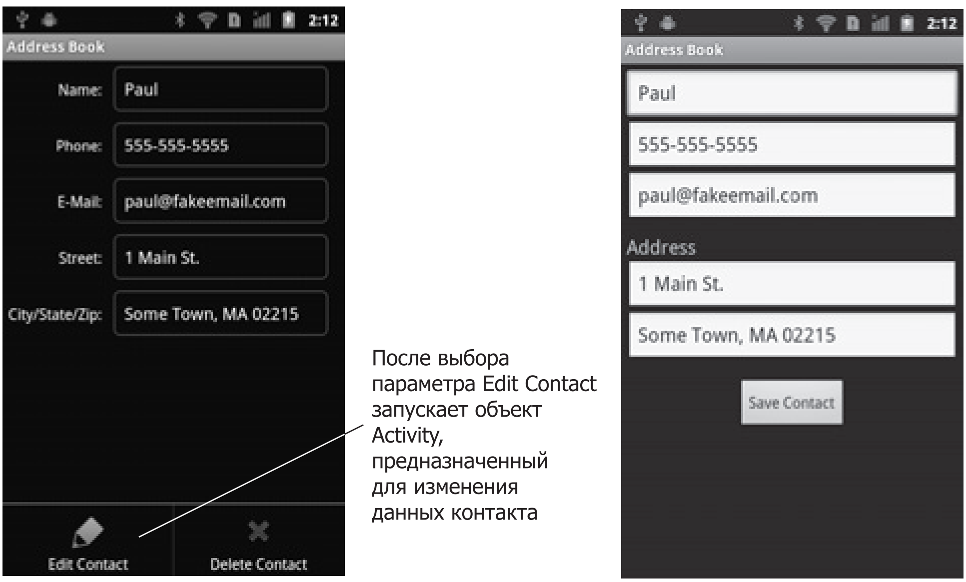


Рис. 10.2. Изменение данных контакта

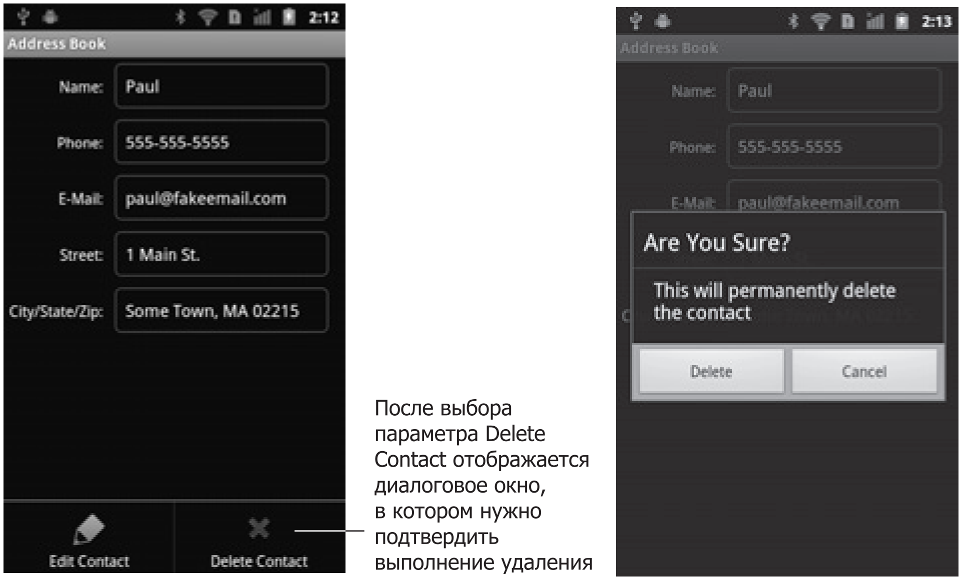


Рис. 10.3. Удаление контакта из базы данных

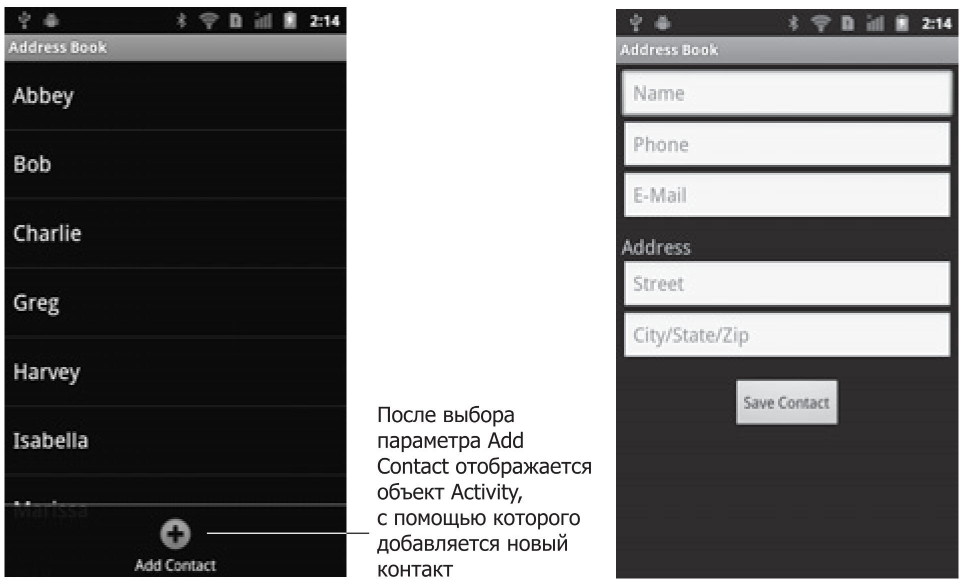


Рис. 10.4. Добавление контакта в базу данных

Добавление контакта

После первого запуска приложения список контактов пуст. Коснитесь кнопки меню устройства, затем выберите параметр Add Contact (Добавить контакт). Появится экран, с помощью которого можно добавить новую запись в список контактов. После ввода информации о контакте коснитесь кнопки Save Contact (Сохранить контакт) для сохранения контакта в базе данных и возврата на главный экран приложения. Если вы передумали добавлять контакт в базу данных, коснитесь кнопки Back (Назад) устройства для возврата к главному экрану. При необходимости добавьте дополнительные контакты.

Просмотр контакта

Для просмотра дополнительных сведений о контакте, добавленном в список, коснитесь его имени.

Изменение контакта

Во время просмотра сведений о контакте коснитесь кнопки меню устройства, затем выберите параметр Edit Contact (Изменить контакт). Появится экран с компонентами EditTexts, которые уже заполнены данными контакта. При необходимости измените эти данные, затем коснитесь кнопки Save Contact (Сохранить контакт) для сохранения обновленной информации в базе данных и возврата на главный экран приложения.

Удаление контакта

В процессе просмотра подробных сведений о контакте коснитесь кнопки меню устройства, затем выберите параметр Delete Contact (Удалить контакт). Если вы действительно хотите удалить контакт, подтвердите операцию удаления в появившемся диалоговом окне. После этого контакт будет удален из базы данных, а приложение вернется к главному экрану.

Оверскроллинг в Android 2.3

Как и в Android 2.3, списки, подобные используемым для отображения контактов в приложении, поддерживают *оверскроллинг*. Этот визуальный эффект (выделяется оранжевой подсветкой) позволяет определить момент достижения верхней или нижней части списка во время выполнения прокрутки его содержимого. Оранжевая подсветка появляется после попытки выполнить прокрутку за пределы списка.

10.3. Обзор применяемых технологий

В этом разделе вашему вниманию будут представлены новые технологии, применяемые в процессе создания приложения Address Book (по мере их использования в главе).

Определение дополнительных элементов Activity в манифесте приложения

В файле AndroidManifest.xml описаны компоненты приложения. В предыдущих главах для каждого приложения использовался единственный класс Activity. В приложении из этой главы используются три класса Activity. Каждый класс Activity должен быть описан в манифесте приложения (раздел 10.4.2).

Определение и применение стилей к компонентам GUI

Сначала необходимо определить пары «атрибут-значение» для общих компонентов GUI в виде ресурсов стилей XML (раздел 10.4.3). Затем эти стили применяются ко

всем компонентам, которые разделяют соответствующие значения (раздел 10.4.6). При этом используется атрибут style. Произвольные изменения, внесенные в стиль, будут автоматически применены ко всем компонентам GUI, использующим этот стиль.

Определение фона для компонентов TextView

По умолчанию у компонентов TextView отсутствует граница. Чтобы задать границу, укажите значение Drawable для атрибута android:background компонента TextView. В качестве значения Drawable может использоваться изображение, хотя для этого приложения можно определить новый тип Drawable, используя XML-представление фигуры (раздел 10.4.4). XML-файл, включающий ресурс Drawable, может быть помещен в папку drawable, которую следует создать в папке res приложения.

Определение формата элементов ListView

Это приложение использует компонент ListView (пакет android.widget) для отображения списка контактов в виде списка элементов, который можно прокручивать в случае, если весь список не помещается на экране. Можно определить ресурс разметки (раздел 10.4.5), который будет применяться для отображения каждого компонента ListView.

Создание XML-ресурсов и их «раздувание» с помощью MenuInflater

В предыдущих приложениях, использующих меню, компоненты MenuItems создавались программно. В этом приложении определения компонентов MenuItems хранятся в формате XML, затем они программно «раздуваются» (разделы 10.5.1 и 10.5.2). При этом используется MenuInflater класса Activity (пакет android.view), который подобен LayoutInflater. Для улучшения визуального восприятия элементов меню дополнительно используются некоторые из стандартных пиктограмм Android.

Расширение класса ListActivity для создания класса Activity, включающего компонент ListView

Если основная задача класса Activity заключается в отображении прокручиваемого списка элементов, можно расширить класс ListActivity (пакет android.app, раздел 10.5.1), который использует компонент ListView, занимающий весь экран (стандартный макет экрана). Класс ListView является подклассом класса AdapterView (пакет android.widget) — компонент GUI, связанный с источником данных с помощью объекта Adapter (пакет android.widget). В данном приложении используется класс CursorAdapter (пакет android.widget) для отображения результатов запроса к базе данных с помощью ListView.

С помощью класса Adapter можно связывать с данными несколько типов AdapterView. Дополнительные сведения о связывании данных в Android и ряд руководств можно найти на веб-сайте developer.android.com/guide/topics/ui/binding.html.

Использование явно определенных объектов для запуска другого класса Activity в том же самом приложении и передачи данных этому же классу Activity

Пользователь разрабатываемого в этой главе приложения может просматривать текущий контакт, добавлять новый контакт либо изменять существующий контакт. Для выполнения каждой из перечисленных операций запускается определенный класс Activity. В главе 5 было продемонстрировано использование неявным образом

определенного объекта Intent для отображения URL-ссылки в окне веб-браузера устройства. В разделах 10.5.1 и 10.5.2 будет продемонстрировано использование явным образом определенных объектов Intent для запуска альтернативного объекта Activity в текущем приложении и передача данных из одного класса Activity в другой. В разделе 10.5.3 будет показано, каким образом вернуться к классу Activity, который был запущен из определенного класса Activity.

Работа с базой данных SQLite

Информация о контактах приложения хранится в базе данных SQLite. Система управления базами данных SQLite (www.sqlite.org) — наиболее популярная СУБД во всем мире. Классы Activity, используемые в приложении, взаимодействуют с базой данных SQLite с помощью класса утилиты DatabaseConnector (раздел 10.5.4). Внутри этого класса можно использовать вложенный подкласс класса SQLiteOpenHelper (пакет android.database.sqlite), который упрощает создание базы данных и позволяет воспользоваться объектом SQLiteDatabase (пакет android.database.sqlite) для работы с содержимым базы данных. Для управления результатами запроса к базе данных используется класс Cursor (пакет android.database).

Использование многопоточности для выполнения операций с базой данных за пределами потока GUI

В процессе выполнения долгих операций применяется блокировка выполнения до завершения этих операций за пределами потока GUI (то есть блокировка доступа к файлам и базе данных). В результате облегчается поддержка «отзывчивости» приложений и исключается появление диалоговых окон Activity Not Responding (ANR, Деятельность не отвечает). Эти окна отображаются в том случае, если Android «думает», что GUI «не отвечает». Если нужно получить доступ к результатам выполнений операций с базами данных в потоке GUI, используется класс AsyncTask (пакет android.os) для выполнения операций в одном потоке и получения результатов выполнения в потоке GUI. Детали создания и манипулирования потоками реализуются с помощью класса AsyncTask, причем результаты манипулирования передаются из класса AsyncTask в поток GUI.

10.4. Создание графического интерфейса пользователя и файлов ресурсов

В этом разделе будут созданы файлы ресурсов Address Book и файлы разметки GUI. Для экономии места мы не будем приводить здесь файл ресурса strings.xml приложения и файлы разметки для класса ViewContact из Activity (view_contact.xml) и класса AddEditContact (add_contact.xml). Чтобы просмотреть содержимое этих файлов, откройте их в проекте в Eclipse.

10.4.1. Создание проекта

Начните с создания нового проекта Android под названием AddresBook. В диалоговом окне New Android Project (Новый проект Android) укажите следующие значения, затем нажмите кнопку Finish (Готово):

- Build Target (Операционная система): Android 2.3.3.
- Application name (Имя приложения): Address Book.

- Package name (Название пакета): com.deitel.adressbook.
- Create Activity (Создать деятельность): AdressBook.
- Min SDK Version (Минимальная версия SDK): 8.

10.4.2. Файл AndroidManifest.xml

В листинге 10.1 приведен код файла AndroidManifest.xml для этого приложения, который включает элемент activity для каждого класса Activity в приложении. В строке 14–15 определяется элемент activity для класса AddEditContact. В строках 16–17 определяется элемент activity класса ViewContact.

Листинг 10.1. Файл AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android=http://schemas.android.com/apk/res/android
3     package="com.deitel.addressbook" android:versionCode="1"
4     android:versionName="1.0">
5     <application android:icon="@drawable/icon"
6        android:label="@string/app_name">
7        <activity android:name=".AddressBook"
8           android:label="@string/app_name">
9           <intent-filter>
10             <action android:name="android.intent.action.MAIN" />
11             <category android:name="android.intent.category.LAUNCHER" />
12          </intent-filter>
13       </activity>
14       <activity android:name=".AddEditContact"
15          android:label="@string/app_name"></activity>
16       <activity android:name=".ViewContact"
17          android:label="@string/app_name"></activity>
18    </application>
19    <uses-sdk android:minSdkVersion="8" />
20 </manifest>
```

10.4.3. Файл styles.xml

В листинге 10.2 определяются ресурсы стилей, применяемые в файле разметки view_contact.xml (раздел 10.4.6). Подобно XML-документам, представляющим другие значения, XML-документ, включающий элементы style, находится в папке res/values приложения. Каждый элемент style определяет элемент name, см. строку 3, используемый для применения стиля к одному или нескольким компонентам GUI, а также один или несколько элементов item (см. строку 4), каждый из которых определяет применяемые атрибуты XML «имя» и «значение».

Листинг 10.2. Стили, определенные в файле styles.xml и находящиеся в папке res/values приложения

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
```

продолжение ⇨

Листинг 10.2 *(продолжение)*

```
 3    <style name="ContactLabelTextView">
 4       <item name="android:layout_width">wrap_content</item>
 5       <item name="android:layout_height">wrap_content</item>
 6       <item name="android:gravity">right</item>
 7       <item name="android:textSize">14sp</item>
 8       <item name="android:textColor">@android:color/white</item>
 9       <item name="android:layout_marginLeft">5dp</item>
10       <item name="android:layout_marginRight">5dp</item>
11       <item name="android:layout_marginTop">5dp</item>
12    </style>
13    <style name="ContactTextView">
14       <item name="android:layout_width">wrap_content</item>
15       <item name="android:layout_height">wrap_content</item>
16       <item name="android:textSize">16sp</item>
17       <item name="android:textColor">@android:color/white</item>
18       <item name="android:layout_margin">5dp</item>
19       <item name="android:background">@drawable/textview_border</item>
20    </style>
21 </resources>
```

10.4.4. Файл textview_border.xml

Стиль ContactTextView, определенный в листинге 10.2 (строки 13–20), задает видимость компонентов TextView, используемых для отображения деталей контакта с помощью класса ViewContact класса Activity. В строке 19 определяется значение Drawable в качестве значения атрибута android:background класса TextView. Используемое здесь значение Drawable (файл textview_border) определено в XML в качестве элемента shape (рис. 10.3) и хранится в папке приложения res/drawable. Атрибут shape элемента android:shape (строка 3) может принимать значение "rectangle" (используется в примере), "oval", "line" или "ring". Элемент corners (строка 4) задает радиус скругленных углов прямоугольника. Элемент stroke (строка 5) определяет толщину и цвет линии прямоугольника. Элемент padding (строки 6–7) — отступ вокруг содержимого элемента, к которому применяется Drawable. Следует определить величины отступа сверху, слева, справа и снизу отдельно. Полную спецификацию по определению форм в XML можно найти на веб-сайте: developer.android.com/guide/topics/resources/ drawable-resource.html#Shape.

Листинг 10.3. Представление XML для Drawable, используемое для создания границы вокруг компонента TextView

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <shape xmlns:android=http://schemas.android.com/apk/res/android
3    android:shape="rectangle" >
4    <corners android:radius="5dp"/>
5    <stroke android:width="1dp" android:color="#555"/>
6    <padding android:top="10dp" android:left="10dp" android:bottom="10dp"
7       android:right="10dp"/>
8 </shape>
```

10.4.5. Файл разметки AddressBook класса Activity: contact_list_item.xml

Класс AddressBook класса Activity расширяет класс ListActivity вместо класса Activity. По умолчанию графический интерфейс пользователя класса ListActivity состоит из компонентов ListView, которые занимают весь экран. При этом не нужно делать отдельную разметку для этого класса Activity. Если нужно настроить графический интерфейс компонента ListActivity, можно определить XML-файл разметки, который должен включать компонент ListView, атрибуту android:id которого присвоено значение "@android:id/list". Этот атрибут подробно рассматривается в главе 12, где разрабатывается приложение Slideshow.

В процессе заполнения данными компонента ListView следует определить формат каждого элемента списка, который определяется с помощью файла разметки contact_list_item.xml (листинг 10.4). Каждый элемент списка включает одно имя контакта, поэтому в разметке определяется единственный компонент TextView, с помощью которого отображается имя. По умолчанию в качестве фонового цвета компонента ListView используется черный цвет, поэтому мы задаем белый цвет (строка 5). С помощью атрибута android:id выполняется связывание данных с компонентом TextView. В строке 6 переменной listPreferredItemHeight присваивается значение минимальной высоты элемента списка (встроенная константа атрибута Android). В строке 7 выбирается выравнивание элемента списка center_vertical. Если элемент списка должен состоять из нескольких частей, в разметке list-item следует использовать несколько элементов, причем для каждого из них требуется атрибут android:id. Дополнительные сведения относительно использования атрибутов android:id приведены в разделе 10.5.1. На рис. 10.1 показан внешний вид элементов list-item.

Листинг 10.4. Разметка для каждого элемента встроенного компонента ListView класса AddressBook объекта ListActivity

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <TextView xmlns:android=http://schemas.android.com/apk/res/android
3      android:id="@+id/contactTextView" android:layout_width="match_parent"
4      android:layout_height="wrap_content" android:padding="8dp"
5      android:textSize="20sp" android:textColor="@android:color/white">
6      android:minHeight="?android:attr/listPreferredItemHeight"
7      android:gravity="center_vertical"></TextView>
```

10.4.6. Разметка для класса ViewContact класса Activity: view_contact.xml

Если пользователь выбирает контакт в AddressBook Activity, приложение запускает ViewContact Activity (рис. 10.5). В разметке этого класса Activity (файл view_contact.xml) применяется компонент ScrollView, включающий компонент TableLayout, в котором каждый компонент TableRow содержит два компонента TextView. Единственное новое свойство этой разметки заключается в том, что ко всем компонентам TextView применены стили, определенные в листинге 10.6. Например, в строках 11–15 файла разметки содержатся следующие стили:

```
<TextView android:id="@+id/nameLabelTextView"
   style="@style/ContactLabelTextView"
   android:text="@string/label_name"></TextView>
<TextView android:id="@+id/nameTextView"
   style="@style/ContactTextView"></TextView>
```

Они применяются к компонентам TextViews в первом компоненте TableRow. Каждый компонент TextView использует атрибут style для определения применяемого стиля. При этом используется синтаксис @style/styleName.

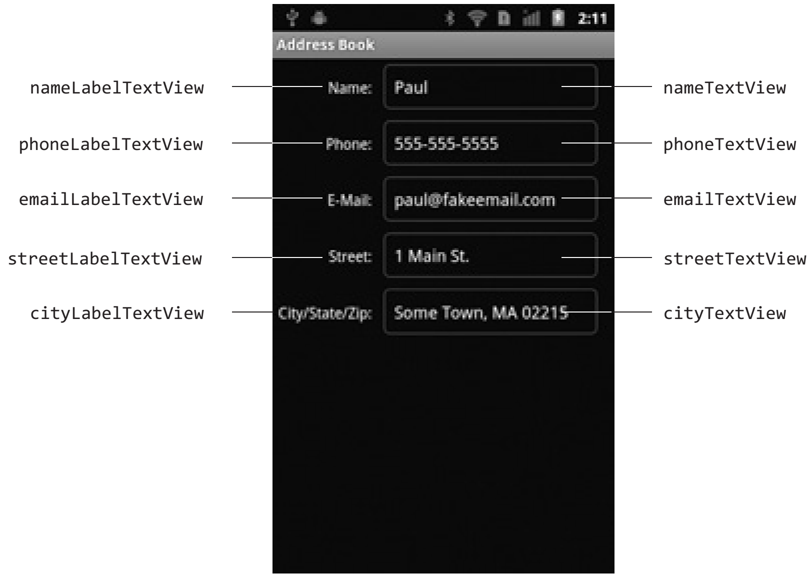


Рис. 10.5. Компоненты GUI класса ViewContact, относящегося к классу Activity, которые подписаны в соответствии со значениями свойства id. В качестве основного компонента GUI выступает компонент ScrollView, включающий компонент TableLayout с пятью компонентами TableRows

10.4.7. Разметка для класса AddEditContact класса Activity: файл add_contact.xml

Если пользователь выбирает элемент меню Add Contact из объекта AddressBook класса Activity либо элемент меню Edit Contact из объекта ViewContact класса Activity, приложение запускает объект AddEditContact класса Activity (рис. 10.6). В разметке Activity используется компонент ScrollView, включающий вертикальный макет LinearLayout. Если класс Activity запущен из класса AddressBook, относящегося к Activity, компоненты EditText будут «обнулены», и отобразятся подсказки (определены в строках 12, 17, 22, 33 и 38 XML-файла разметки). В противном случае компоненты EditTexts отобразят данные контакта, которые будут переданы классу AddEditContact Activity класса ViewContact Activity. Каждый компонент EditText определяет атрибуты android:inputType и android:imeOptions. Для устройств, которые отображают программную клавиатуру, атрибут android:inputType (в строках 13, 18, 23, 34 и 39 XML-файла разметки) определяет

клавиатуру, которая отображается после касания пользователем соответствующего компонента EditText. С помощью этого атрибута можно настроить клавиатуру в соответствии с типом данных, которые пользователь должен ввести в соответствующий EditText. Как и в главе 5, с помощью атрибута android:imeOptions отображается кнопка Next (Далее) на программных клавиатурах для компонентов nameEditText, emailEditText, phoneEditText и streetEditText. Если один из этих компонентов получает фокус, касание соответствующей кнопки Button приведет к передаче фокуса следующему компоненту EditText. Если компонент cityEditText получает фокус, можно скрыть программную клавиатуру, коснувшись кнопки Done (Готово) клавиатуры.

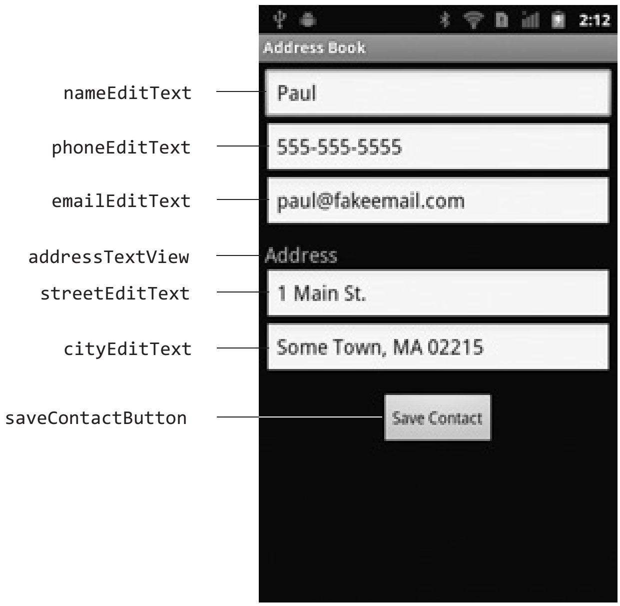


Рис. 10.6. Компоненты GUI класса AddEditContact Activity, помеченные с помощью значений свойства id. В качестве основного компонента GUI выступает компонент ScrollView, который включает вертикальный макет LinearLayout

10.4.8. Определение компонентов MenuItems приложения с помощью ресурсов меню, заданных в XML-формате

В листингах 10.5 и 10.6 определяются ресурсы меню для AddressBook Activity и ViewContact Activity соответственно. Файлы ресурсов, которые определяют меню, находятся в папке res/menu приложения (эту папку нужно создать) и добавляются в проект подобно другим файлам ресурсов (изначально описаны в разделе 3.5). В диалоговом окне New Android XML File (Новый XML-файл Android) можно выбрать параметр Menu (Меню) в качестве типа ресурса. Каждый XML-файл ресурса меню содержит элемент root menu, вложенный в элементы item, которые представляет каждый компонент MenuItem. В разделах 10.5.1 и 10.5.2 будет показано, каким образом можно «раздувать» меню.

Листинг 10.5. Ресурс меню AddressBook класса Activity

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android">
3     <item android:id="@+id/addContactItem"
4        android:title="@string/menuitem_add_contact"
5        android:icon="@android:drawable/ic_menu_add"
6        android:titleCondensed="@string/menuitem_add_contact"
7        android:alphabeticShortcut="e"></item>
8  </menu>
```

Листинг 10.6. Ресурс меню ViewContact класса Activity

```
 1  <?xml version="1.0" encoding="utf-8"?>
 2  <menu xmlns:android="http://schemas.android.com/apk/res/android">
 3     <item android:id="@+id/editItem"
 4        android:title="@string/menuitem_edit_contact"
 5        android:orderInCategory="1" android:alphabeticShortcut="e"
 6        android:titleCondensed="@string/menuitem_edit_contact"
 7        android:icon="@android:drawable/ic_menu_edit"></item>
 8     <item android:id="@+id/deleteItem"
 9        android:title="@string/menuitem_delete_contact"
10        android:orderInCategory="2" android:alphabeticShortcut="d"
11        android:titleCondensed="@string/menuitem_delete_contact"
12        android:icon="@android:drawable/ic_delete"></item>
13   </menu>
```

Для каждого элемента меню определяется значение атрибута android:id, обеспечивающего возможность взаимодействия с соответствующим компонентом MenuItem программным образом. Другие атрибуты элементов меню приведены в следующем списке:

- Атрибуты android:title и android:titleCondensed. С помощью этих атрибутов определяется текст, отображаемый компонентов MenuItem. Второй атрибут применяется в том случае, если обычный заголовок слишком широкий и не «помещается» в выбранном элементе меню.
- Атрибут android:icon. Этот атрибут определяет значение Drawable, которое отображается компонентом MenuItem над текстом заголовка. В рассматриваемом примере компонентов MenuItems используются три стандартных пиктограммы, которые поддерживаются Android SDK. Эти пиктограммы находятся в папках SDK platforms в каждой папке data/res/drawable-hdpi, соответствующей версии платформы. Чтобы сослаться на эти пиктограммы в разметке XML, используйте в ссылке префикс @android:drawable/имя_пиктограммы (см. листинг 10.5, строка 5, а также листинг 10.6, строки 7 и 12).
- Атрибут android:alphabeticShortcut. С помощью этого атрибута определяется буква, которую должен нажать пользователь на физической клавиатуре, чтобы выбрать элемент меню.
- Атрибут android:orderInCategory. Данный атрибут определяет порядок, в котором отображаются элементы меню MenuItems. Этот атрибут не используется в листинге 10.5, поскольку используется лишь один элемент меню MenuItem.

Чтобы получить подробные сведения о ресурсах меню, обратитесь к веб-сайту developer.android.com/guide/topics/resources/menu-resource.html.

10.5. Создание приложения

Это приложение состоит из четырех классов — класс AddressBook (подкласс ListActivity, листинги 10.7–10.12), класс ViewContact (листинги 10.13–10.17), класс AddEditContact (листинги 10.18–10.21) и класс DatabaseConnector (листинги 10.22–10.25). Как и при разработке предыдущих приложений, основной класс Activity приложения (класс AddressBook) создается при формировании проекта, но его нужно изменить, чтобы расширить класс ListActivity. Следует добавить другие классы Activity и класс DatabaseConnector в папку проекта src/com.deitel.addressbook.

10.5.1. Подкласс AddressBook класса ListActivity

Класс AddressBook (см. листинги 10.7–10.12) поддерживает набор функций для первого класса Activity, отображаемого приложением. Как отмечалось ранее в главе, класс расширяет класс ListActivity, а не данный класс Activity, поскольку главная задача этого класса Activity заключается в отображении компонента ListView, включающего контакты пользователя.

Операторы package, import и переменные экземпляра класса

В листинге 10.7 перечислены операторы package, import и переменные экземпляра класса AddressBook. Операции импорта, применяемые для создания новых классов, были рассмотрены в разделе 10.3. Константа ROW_ID применяется в качестве ключа в паре «ключ–значение», который передается между деятельностями (листинг 10.12). Переменная экземпляра класса contactListView будет ссылаться на встроенный в класс AddressBook компонент ListView, обеспечивая возможность программного взаимодействия с этим компонентом. Переменная экземпляра класса contactAdapter будет ссылаться на объект CursorAdapter, который заполняет компонент ListView класса AddressBook.

Листинг 10.7. Операторы package, import и переменные экземпляра класса AddressBook

```
// AddressBook.java
// Основной класс Activity для приложения Address Book.
package com.deitel.addressbook;

import android.app.ListActivity;
import android.content.Intent;
import android.database.Cursor;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
```

продолжение ☞

Листинг 10.7 *(продолжение)*

```
import android.widget.AdapterView.OnItemClickListener;
import android.widget.CursorAdapter;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;

public class AddressBook extends ListActivity
{
   public static final String ROW_ID = "row_id";  // дополнительный
                                                  // ключ Intent
   private ListView contactListView;     // компонент ListView
                                         // из ListActivity
   private CursorAdapter contactAdapter; // адаптер для ListView

```

Переопределение метода onCreate класса Activity

Метод onCreate (см. листинг 10.8, строки 26–32) инициализирует класс Activity. Ранее уже упоминалось о том, что класс ListActivity уже включает компонент ListView, который занимает все пространство Activity. В связи с этим не нужно «раздувать» GUI с помощью метода setContentView, как это делалось при разработке предыдущих приложений. В строке 31 применяется унаследованный метод getListView класса ListActivity для получения ссылки на встроенный компонент ListView. Затем в строке 32 OnItemClickListener из компонента ListView присваивается значение viewContactListener (см. листинг 10.12) в ответ на выбор пользователем одного из элементов ListView.

Листинг 10.8. Переопределение метода onCreate класса Activity

```
26   // вызывается при первом создании деятельности
27   @Override
28   public void onCreate(Bundle savedInstanceState)
29   {
30      super.onCreate(savedInstanceState); // вызывает onCreate
                                            // из суперкласса
31       contactListView = getListView(); // доступ к встроенному ListView
32       contactListView.setOnItemClickListener(viewContactListener);
33
34       // отображение имени каждого контакта на TextView
         // в разметке ListView
35       String[] from = new String[] { "name" };
36       int[] to = new int[] { R.id.contactTextView };
37       CursorAdapter contactAdapter = new SimpleCursorAdapter(
38          AddressBook.this, R.layout.contact_list_item, null, from, to);
39       setListAdapter(contactAdapter); // настройка адаптера contactView
40   }   // конец описания метода onCreate
41
```

Чтобы отобразить результаты из класса Cursor с помощью компонента ListView, создается новый объект CursorAdapter (строки 35–38). Этот объект отображает данные из Cursor в форме, которая используется в элементе управления ListView. Класс SimpleCursorAdapter — это подкласс класса CursorAdapter, который был разработан для облечения

отображения столбцов класса Cursor непосредственно на компоненты TextViews или ImagesView, определенные в XML-разметке. Для создания класса SimpleCursorAdapter сначала определите массивы, включающие имена столбцов, для отображения на компоненты GUI. Также следует определить ID ресурсов для компонентов GUI, которые отображают данные из именованных столбцов. В строке 35 создается массив String, показывающий, что может отображаться лишь именованный столбец. В строке 36 создается параллельный массив типа int, содержащий ID ресурсов для соответствующих компонентов GUI (в рассматриваемом случае, R.id.contactTextView). В строках 37–38 создается класс SimpleCursorAdapter. Конструктор этого класса получает следующие параметры:

- параметр Context, в котором выполняется компонент ListView (то есть AddressBook Activity);
- ID ресурса для разметки, который используется для отображения каждого элемента в ListView;
- класс Cursor, который обеспечивает доступ к данным, — этому аргументу присваивается значение null, поскольку класс Cursor определяется позже;
- массив String, включающий отображаемые названия столбцов;
- массив типа int, включающий идентификаторы ID соответствующих ресурсов GUI.

В строке 39 используется унаследованный метод setListAdapter класса ListActivity для связывания классов ListView и CursorAdapter. В результате ListView может отображать данные.

Переопределение методов onResume и onStop класса Activity

Как упоминалось в разделе 8.5.1, метод onResume (см. листинг 10.9, строки 42–49) вызывается при каждом возвращении Activity на «передний план», включая время первого создания класса Activity. В данном приложении метод onResume создает и вызывает метод AsyncTask (строка 48) типа GetContactsTask (определен в листинге 10.10), который получает полный список контактов из базы данных и настраивает класс Cursor из contactAdapter таким образом, чтобы вводить данные в компонент ListView класса AddressBook. Метод AsyncTask инициирует выполнение задачи в отдельном потоке. В этом случае применение аргумента execute метода говорит о том, что при выполнении задачи не принимаются какие-либо аргументы. Этот метод может принимать различное число аргументов, которые, в свою очередь, передаются как аргументы методу задания doInBackground. При каждом выполнении оператора в строке 48 создается новый объект GetContactsTask. Создание этого объекта обязательно, поскольку каждый метод AsyncTask может быть вызван *один раз*.

Листинг 10.9. Переопределение методов onResume и onStop класса Activity

```
42   @Override
43   protected void onResume()
44   {
45      super.onResume(); // вызывает метод onResume суперкласса
46
```

продолжение ⇨

Листинг 10.9 *(продолжение)*

```
47     // создает новый объект GetContactsTask и вызывает его
48     new GetContactsTask().execute((Object[]) null);
49  }  // конец описания метода onResume
50
51  @Override
52  protected void onStop()
53  {
54     Cursor cursor = contactAdapter.getCursor(); // доступ
                                                  // к текущему Cursor
55
56     if (cursor != null)
57        cursor.deactivate(); // деактивирование
58
59     contactAdapter.changeCursor(null); // адаптировано к отсутствию
                                          // Cursor
60     super.onStop();
61  }  // конец описания метода onStop
62
```

Метод onStop класса Activity (см. листинг 10.9, строки 51–61) вызывается, если класс Activity уже не видим пользователю, — обычно это происходит в том случае, если запущен или возвращается на передний план другой класс Activity. В рассматриваемом случае класс Cursor, с помощью которого вводятся данные в ListView, не нужен. Поэтому в строке 54 вызывается метод getCursor класса CursorAdapter, с помощью которого вызывается текущий класс Cursor из contactAdapter. Затем в строке 57 вызывается метод deactivate класса Cursor, с помощью которого освобождаются ресурсы, используемые классом Cursor. Затем в строке 59 вызывается метод changeCursor класса CursorAdapter с аргументом null для удаления Cursor из класса CursorAdapter.

Подкласс GetContactsTask класса AsyncTask

Вложенный класс GetContactsTask (см. листинг 10.10) расширяет класс AsyncTask. Класс определяет порядок взаимодействия с базой данных для получения имен контактов и возвращения результатов в поток GUI класса Activity для их отображения с помощью ListView. Класс AsyncTask имеет обобщенный тип, который требует три параметра:

- Первый параметр представляет собой тип списка переменной длины метода doInBackground класса AsyncTask (строки 50–57). Если вызывается метод execute класса AsyncTask, метод doInBackground задания выполняет задание в отдельном потоке выполнения. В рассматриваемом случае метод doInBackground не требует дополнительных данных для выполнения задачи, поэтому мы определили Object в качестве параметра типа и передали null в качестве аргумента методу execute класса AsyncTask, а затем вызвали метод doInBackground.
- Второй параметр представляет собой тип списка переменной длины для метода onProgressUpdate класса AsyncTask. Этот метод вызывается в потоке GUI и используется для получения промежуточных обновлений определенного типа из длительно выполняющейся задачи. В рассматриваемом примере мы не используем это свойство, поэтому указываем тип Object и игнорируем этот параметр типа.

- Третий параметр представляет тип возвращаемого результата выполняемой задачи, который передается методу onPostExecute класса AsyncTask (строки 80–85). Этот метод выполняется в потоке GUI и позволяет классу Activity использовать результаты класса AsyncTask.

Листинг 10.10. Подкласс GetContactsTask класса AsyncTask

```
63  // выполнение запроса к базе данных за пределами потока GUI
64  private class GetContactsTask extends AsyncTask<Object, Object, Cursor>
65  {
66     DatabaseConnector databaseConnector =
67        new DatabaseConnector(AddressBook.this);
68
69     // выполнение доступа к базе данных
70     @Override
71     protected Cursor doInBackground(Object... params)
72     {
73        databaseConnector.open();
74
75        // доступ к курсору, включая вызов контактов
76        return databaseConnector.getAllContacts();
77     }  // конец определения метода doInBackground
78
79     // использование Cursor, возвращенного методом doInBackground
80     @Override
81     protected void onPostExecute(Cursor result)
82     {
83        contactAdapter.changeCursor(result); // установка Cursor адаптера
84        databaseConnector.close();
85     }  // конец определения метода onPostExecute
86  }  // конец определения класса GetContactsTask
87
```

Основное преимущество использования класса AsyncTask заключается в том, что этот класс обрабатывает все, что связано с созданием потоков и вызовов соответствующих методов в соответствующих пользовательских потоках, избавляя пользователя от необходимости непосредственно взаимодействовать с механизмом управления потоками.

В строках 66–67 создается новый объект класса утилиты DatabaseConnector, в качестве аргумента конструктору класса передается объект Context (AddressBook.this). (Класс DatabaseConnector будет подробно рассмотрен в разделе 10.5.4.)

Метод doInBackground (строки 70–77) использует класс databaseConnector для открытия подключения к базе данных и получения всех контактов из базы данных. Класс Cursor, возвращаемый getAllContacts, передается методу onPostExecute (строки 80–86). Этот метод получает класс Cursor, содержащий результаты, и передает его методу changeCursor класса CursorAdapter. В результате компонент ListView класса Activity может получать данные самостоятельно.

Управление курсорами

В текущем классе Activity можно управлять курсорами с помощью различных методов Cursor и CursorAdapter. Класс Activity может также управлять пользовательскими

курсорами. Метод startManagingCursor класса Activity определяет для класса Activity управление жизненным циклом Cursor на основе жизненного цикла класса Activity. Если класс Activity остановлен, он может вызвать метод deactivate для любых управляемых в настоящее время курсоров. Если восстанавливается класс Activity, направляется повторный запрос курсоров. Если класс Activity разрушен, он автоматически вызовет метод close, с помощью которого освобождаются все ресурсы, выделенные для всех управляемых курсоров. Деактивированный класс Cursor потребляет меньше ресурсов, чем активный, поэтому следует сравнять жизненный цикл класса Cursor с родительским Activity, если Cursor не используется вместе с другими обьектами Activity. Позволяя классу Activity управлять жизненным циклом Cursor, вы гарантируете, что Cursor будет закрыт, когда он больше не нужен.

Переопределение методов onCreateOptionsMenu и onOptionsItemSelected класса Activity

Если пользователь открывает меню Activity, метод onCreateOptionsMenu (см. листинг 10.11, строки 89–96) с помощью MenuInflater создает меню на основе файла addressbook_menu.xml, который включает Add Contact MenuItem. Доступ к MenuInflater можно получить путем вызова метода getMenuInflater класса Activity. После выбора пользователем компонента MenuItem метод onOptionsItemSelected (строки 99–107) запускает метод AddEditContact класса Activity (раздел 10.5.3). В строках 103–104 создается новый явно заданный класс Intent, предназначенный для запуска этого класса Activity. Используемый здесь конструктор класса Intent получает объект Context, из которого может быть запущен класс Activity, и класс, который представляет запускаемый класс Activity (AddEditContact.class). Затем объект Intent передает унаследованный метод startActivity класса Activity, применяемый для запуска этого класса Activity.

Листинг 10.11. Переопределенные методы onCreateOptionsMenu и onOptionsItemSelected класса Activity

```
 88 // создание меню Activity на основе XML-файла ресурсов меню
 89 @Override
 90 public boolean onCreateOptionsMenu(Menu menu)
 91 {
 92    super.onCreateOptionsMenu(menu);
 93    MenuInflater inflater = getMenuInflater();
 94    inflater.inflate(R.menu.addressbook_menu, menu);
 95    return true;
 96 }  // конец определения метода onCreateOptionsMenu
 97
 98 // обработка варианта, выбранного из меню параметров
 99 @Override
100 public boolean onOptionsItemSelected(MenuItem item)
101 {
102    // создание нового объекта Intent для запуска метода
       // AddEditContact класса Activity
103    Intent addNewContact =
104       new Intent(AddressBook.this, AddEditContact.class);
105    startActivity(addNewContact); // start the AddEditContact Activity
```

```
106         return super.onOptionsItemSelected(item); // вызов метода
                                                      // суперкласса
107   }  // конец определения метода onOptionsItemSelected
108
```

Анонимный внутренний класс, реализующий интерфейс OnItemClickListener для обработки событий ListView

Интерфейс viewContactListener класса OnItemClickListener (см. листинг 10.12) запускает метод ViewContact класса Activity для отображения выбранного пользователем контакта. Метод onItemClick получает следующие параметры:

- ссылка на объект AdapterView, с которым взаимодействует пользователь (то есть на компонент ListView);
- ссылка на корневой компонент View для выбранного пользователем элемента меню;
- индекс выбранного элемента списка для ListView;
- уникальный идентификатор ID типа long для выбранного элемента — в рассматриваемом случае подразумевается ID в классе Cursor.

Листинг 10.12. Интерфейс OnItemClickListener класса viewContactListener, обрабатывающий события, связанные с выбором компонентов ListView

```
109   // слушатель события, которое отвечает на выбор имени контакта
110   // в ListView
111   OnItemClickListener viewContactListener = new OnItemClickListener()
112   {
113      @Override
114      public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
115         long arg3)
116         {
117            // создание Intent для запуска ViewContact Activity
118            Intent viewContact =
119               new Intent(AddressBook.this, ViewContact.class);
120
121            // передача ID строки выбранного контакта в качестве
               // расширения объекта Intent
122            viewContact.putExtra(ROW_ID, arg3);
123            startActivity(viewContact); // запуск ViewContact Activity
124         }  // конец описания метода onItemClick
125      }; // конец описания viewContactListener
126   }  // конец описания класса AddressBook
```

В строках 118–119 создается определенный явно класс Intent, применяемый для запуска ViewContact класса Activity. Для отображения соответствующего контакта класс ViewContact класса Activity должен «знать», какую запись нужно выбрать в данный момент времени. Чтобы передавать данные между деятельностями путем добавления расширений (extra) в Intent используется метод putExtra класса Intent (строка 122). Этот метод добавляет данные в виде пар ключ–значение в объект Bundle, связанный

с классом Intent. В данном случае пара ключ–значение представляет уникальный идентификатор строки контакта, выбранного пользователем.

10.5.2. Подкласс ViewContact класса Activity

С помощью объекта ViewContact класса Activity (см. листинги 10.13–10.17) отображаются сведения об одном контакте, а также поддерживается меню, с помощью которого пользователь может изменить или удалить контакт.

Операторы package, import и переменные экземпляра класса

В листинге 10.13 приведены операторы package, import и переменные экземпляра класса ViewContact. Операторы import для новых классов были описаны в разделе 10.3. Переменная экземпляра rowID представляет идентификатор уникальной строки контакта в базе данных. Переменные экземпляра класса TextView (строки 20–24) применяются для отображения данных контакта на экране.

Листинг 10.13. Операторы package, import и переменные экземпляра класса ViewContact

```
 1 // ViewContact.java
 2 // Класс Activity, применяемый для просмотра единственного контакта.
 3 package com.deitel.addressbook;
 4 
 5 import android.app.Activity;
 6 import android.app.AlertDialog;
 7 import android.content.DialogInterface;
 8 import android.content.Intent;
 9 import android.database.Cursor;
10 import android.os.AsyncTask;
11 import android.os.Bundle;
12 import android.view.Menu;
13 import android.view.MenuInflater;
14 import android.view.MenuItem;
15 import android.widget.TextView;
16 
17 public class ViewContact extends Activity
18 {
19    private long rowID; // имя выбранного контакта
20    private TextView nameTextView;   // отображает имя контакта
21    private TextView phoneTextView;  // номер телефона контакта
22    private TextView emailTextView;  // e_mail контакта
23    private TextView streetTextView; // номер улицы контакта
24    private TextView cityTextView;   // город/штат/индекс контакта
25 
```

Переопределение методов onCreate и onResume класса Activity

Метод onCreate (см. листинг 10.14, строки 27–43) сначала получает ссылки на компоненты TextView класса Activity, а затем получает ID строки для выбранного контакта. Метод getIntent класса Activity возвращает объект Intent, который запустил Activity. С помощью

этого объекта вызывается метод getExtras класса Intent, который возвращает объект Bundle, включающий произвольные пары ключ–значение, которые были добавлены в класс Intent в виде расширений. Этот метод возвращает значение null, если не были добавлены дополнения. Затем воспользуемся методом getLong класса Bundle для получения целого числа типа long, представляющего ID строки для выбранного контакта.

ПРИМЕЧАНИЕ

Мы не проверяли, что происходит в случае, если значение дополнений равно null (строка 41). Это связано с тем, что в качестве этих значений выступал объект Bundle, создаваемый в приложении. Конечно, тестирование с применением значения null является хорошей практикой, поэтому желательно это сделать. Например, можно зарегистрировать ошибку, а затем вернуть ее из класса Activity путем вызова метода finish.

Метод onResume (строки 46–53) просто создает новый объект AsyncTask типа LoadContactTask (см. листинг 10.15), а затем вызывает его для получения и отображения информации о контакте.

Листинг 10.14. Переопределение метода onCreate класса Activity

```
26  // вызывается после первого создания деятельности
27  @Override
28  public void onCreate(Bundle savedInstanceState)
29  {
30     super.onCreate(savedInstanceState);
31     setContentView(R.layout.view_contact);
32
33     // получение доступа к компонентам EditTexts
34     nameTextView = (TextView) findViewById(R.id.nameTextView);
35     phoneTextView = (TextView) findViewById(R.id.phoneTextView);
36     emailTextView = (TextView) findViewById(R.id.emailTextView);
37     streetTextView = (TextView) findViewById(R.id.streetTextView);
38     cityTextView = (TextView) findViewById(R.id.cityTextView);
39
40     // получение ID строки выбранного контакта
41     Bundle extras = getIntent().getExtras();
42     rowID = extras.getLong("row_id");
43  } // конец определения метода onCreate
44
45  // вызывается при первом создании деятельности
46  @Override
47  protected void onResume()
48  {
49     super.onResume();
50
51     // создание нового класса LoadContactTask и вызов его
52     new LoadContactTask().execute(rowID);
53  } // конец определения метода onResume
54
```

Подкласс GetContactsTask класса AsyncTask

Вложенный класс GetContactsTask (см. листинг 10.15) расширяет класс AsyncTask и определяет порядок взаимодействия с базой данных и получения информации об одном контакте, которая будет отображаться на экране. В этом случае используются три параметра, имеющие обобщенный тип:

- Тип Long применяется для списка параметров переменной длины, передаваемых методу doInBackground класса AsyncTask. Этот список будет включать ID строки, нужный для выборки одного контакта.
- Тип Object применяется для списка параметров переменной длины, передаваемых методу onProgressUpdate класса AsyncTask. Этот тип параметров не применяется в рассматриваемом примере.
- Тип Cursor применяется в качестве типа результатов выполняемой задачи, которые передаются методу onPostExecute класса AsyncTask.

Листинг 10.15. Метод loadContact класса ViewContact

```
// выполняет запрос к базе данных за пределами потока GUI
private class LoadContactTask extends AsyncTask<Long, Object, Cursor>
{
   DatabaseConnector databaseConnector =
      new DatabaseConnector(ViewContact.this);

   // выполнение доступа к базе данных
   @Override
   protected Cursor doInBackground(Long... params)
   {
      databaseConnector.open();

      // получение курсора, содержащего все данные выбранной записи
      return databaseConnector.getOneContact(params[0]);
   }  // конец определения метода doInBackground

   // использование объекта типа Cursor, возвращаемого методом
   // doInBackground
   @Override
   protected void onPostExecute(Cursor result)
   {
      super.onPostExecute(result);

      result.moveToFirst(); // перемещение к первому элементу

      // получение индекса столбца для каждого элемента данных
      int nameIndex = result.getColumnIndex("name");
      int phoneIndex = result.getColumnIndex("phone");
      int emailIndex = result.getColumnIndex("email");
      int streetIndex = result.getColumnIndex("street");
      int cityIndex = result.getColumnIndex("city");

      // заполнение компонентов TextViews выбранными данными
```

```
87         nameTextView.setText(result.getString(nameIndex));
88         phoneTextView.setText(result.getString(phoneIndex));
89         emailTextView.setText(result.getString(emailIndex));
90         streetTextView.setText(result.getString(streetIndex));
91         cityTextView.setText(result.getString(cityIndex));
92
93         result.close(); // закрытие курсора результата
94         databaseConnector.close(); // закрытие подключения к базе данных
95      } // конец определения метода onPostExecute
96   } // конец определения класса LoadContactTask
97
```

В строках 58–59 создается новый объект класса DatabaseConnector (раздел 10.5.4). Метод doInBackground (строки 62–69) открывает подключение к базе данных и вызывает метод getOneContact класса DatabaseConnector, который создает запрос к базе данных на выборку контакта с указанным rowID, переданного в качестве единственного аргумента методу execute класса AsyncTask. Метод doInBackground хранит rowID в массиве params[0].

Результирующий объект Cursor передается методу onPostExecute (строки 72–95). Объект Cursor позиционируется перед первой строкой набора результатов. В рассматриваемом случае набор результатов будет включать единственную запись. Для перемещения объекта Cursor в первую строку набора результатов может использоваться метод moveToFirst (строка 77).

ПРИМЕЧАНИЕ

> Рекомендуется удостовериться в том, что метод moveToFirst класса Cursor возвращает значение true прежде, чем выбираются данные из класса Cursor. В данном приложении в качестве данных выбирается строка из класса Cursor.

Метод getColumnIndex класса Cursor применяется для получения индикаторов столбцов для столбцов в таблице контактов базы данных. (Имена столбцов были жестко закодированы в приложении, но они могут быть реализованы в виде констант String подобно тому, как мы использовали константы ROW_ID в классе AddressBook.) Этот метод возвращает значение -1, если столбец отсутствует в результате запроса. Класс Cursor также поддерживает метод getColumnIndexOrThrow, с помощью которого можно получить исключение в случае, если указанное имя столбца не существует. В строках 87–91 используется метод getString класса Cursor для выборки значений String из столбцов класса Cursor с последующим отображением этих значений в соответствующий компонент TextView. В строках 93–94 закрывается класс Cursor и подключение к базе данных класса Activity, поскольку они больше не требуются. Рекомендуется освобождать ресурсы, такие как подключения к базе данных, если они больше не нужны и могут применяться другими деятельностями.

Переопределение методов onCreateOptionsMenu и onOptionsItemSelected класса Activity

Меню ViewContact класса Activity поддерживает параметры, обеспечивающие изменение и удаление контакта. Метод onCreateOptionsMenu (см. листинг 10.16, строки 99–106) использует метод MenuInflater для создания меню на основе файла ресурсов view_contact.

xml, который включает элементы MenuItems Edit Contact и Delete Contact. Метод onOptionsItemSelected (строки 109–134) использует ID ресурса элемента MenuItem для определения выделенного элемента меню. Если выбран пункт меню Edit Contact, в строках 116–126 создается новый явно определенный класс Intent для компонента AddEditContact класса Activity (раздел 10.5.3), добавляются дополнения в класс Intent, представляющие информацию о контакте, отображаемую в EditTexts компонента AddEditContact класса Activity, а также запускается Activity. Если выбирается элемент меню Delete Contact, в строке 129 вызывается метод утилиты deleteContact (см. листинг 10.17).

Листинг 10.16. Переопределение методов onCreateOptionsMenu и onOptionsItemSelected

```
 98  создание меню Activity на основе XML-файла ресурсов меню
 99  @Override
100  public boolean onCreateOptionsMenu(Menu menu)
101  {
102     super.onCreateOptionsMenu(menu);
103     MenuInflater inflater = getMenuInflater();
104     inflater.inflate(R.menu.view_contact_menu, menu);
105     return true;
106  }  // конец определения метода onCreateOptionsMenu
107
108  // обработка варианта, выбранного среди пунктов меню
109  @Override
110  public boolean onOptionsItemSelected(MenuItem item)
111  {
112     switch (item.getItemId()) // переключение на основе
                                  // выбранного ID MenuItem
113     {
114        case R.id.editItem:
115        // создание Intent для запуска AddEditContact Activity
116        Intent addEditContact =
117           new Intent(this, AddEditContact.class);
118
119        // передача данных выбранного контакта в виде
           // дополнений к Intent
120        addEditContact.putExtra("row_id", rowID);
121        addEditContact.putExtra("name", nameTextView.getText());
122        addEditContact.putExtra("phone", phoneTextView.getText());
123        addEditContact.putExtra("email", emailTextView.getText());
124        addEditContact.putExtra("street", streetTextView.getText());
125        addEditContact.putExtra("city", cityTextView.getText());
126        startActivity(addEditContact); // start the Activity
127        return true;
128     case R.id.deleteItem:
129        deleteContact(); // удаление отображенного контакта
130        return true;
131     default:
132        return super.onOptionsItemSelected(item);
133   } // конец переключателя switch
134 } // конец определения метода onOptionsItemSelected
135
```

Метод deleteContact

Метод deleteContact (см. листинг 10.17) отображает диалоговое окно AlertDialog, которое предлагает пользователю подтвердить удаление текущего выделенного контакта. В этом случае класс AsyncTask удаляет контакт из базы данных SQLite. Если пользователь щелкнет на кнопке Delete Button в диалоговом окне, в строках 153–154 создается новый класс DatabaseConnector. В строках 158–173 создается класс AsyncTask, после вызова которого (строка 176) передается значение типа Long, которое представляет ID строки контакта, методу doInBackground. Этот метод удаляет контакт. В строке 164 вызывается метод deleteContact класса DatabaseConnector, который выполняет фактическое удаление. Как только метод doInBackground завершает выполнение, в строке 171 вызывается метод finish класса Activity для возвращения в класс Activity, который запущен из класса ViewContact, выбранного в классе Activity. В данном случае осуществляется возврат в класс AddressBook класса Activity.

Листинг 10.17. Метод deleteContact класса ViewContact

```
136   // удаление контакта
137   private void deleteContact()
138   {
139      // создание нового AlertDialog Builder
140      AlertDialog.Builder builder =
141         new AlertDialog.Builder(ViewContact.this);
142
143      builder.setTitle(R.string.confirmTitle);     // строка заголовка
144      builder.setMessage(R.string.confirmMessage); // отображаемое
                                                      // сообщение
145
146      // поддержка кнопки OK, которая просто скрывает диалоговое окно
147      builder.setPositiveButton(R.string.button_delete,
148         new DialogInterface.OnClickListener()
149         {
150            @Override
151            public void onClick(DialogInterface dialog, int button)
152            {
153               final DatabaseConnector databaseConnector =
154                  new DatabaseConnector(ViewContact.this);
155
156               // создание класса AsyncTask, удаляющего контакт
157               // из другого потока, после удаления вызывается finish
158               AsyncTask<Long, Object, Object> deleteTask =
159                  new AsyncTask<Long, Object, Object>()
160               {
161               @Override
162               protected Object doInBackground(Long... params)
163               {
164                  databaseConnector.deleteContact(params[0]);
165                  return null;
166               }  // конец определения метода doInBackground
167
168               @Override
```

продолжение ⇨

Листинг 10.17 *(продолжение)*

```
              protected void onPostExecute(Object result)
              {
                 finish(); // возврат в AddressBook Activity
              }  // конец определения метода onPostExecute
           }; // конец определения нового класса AsyncTask

           // вызов класса AsyncTask для удаления контакта с rowID
           deleteTask.execute(new Long[] { rowID });
        }  // конец определения метода onClick
     }  // конец определения анонимного внутреннего класса
  ); // завершение вызова метода setPositiveButton

     builder.setNegativeButton(R.string.button_cancel, null);
     builder.show(); // отображение диалогового окна
  }  // конец определения метода deleteContact
} // конец определения класса ViewContact
```

10.5.3. Подкласс AddEditContact класса Activity

Метод AddEditContact класса Activity (см. листинги 10.18–10.21) позволяет добавлять новый контакт либо изменить информацию существующего контакта.

Операторы package, import и переменные экземпляра класса

В листинге 10.18 находятся операторы package, import и переменные из экземпляра класса AddEditContact. В этом классе Activity не используются новые классы. Переменные экземпляра класса databaseConnector обеспечивают возможность взаимодействия между классом Activity и базой данных. Переменная экземпляра класса rowID представляет текущий контакт, который будет обрабатываться в случае, если этот класс Activity был запущен, чтобы позволить пользователю изменять существующий контакт. Переменные экземпляра класса в строках 20–24 позволяют манипулировать текстом в компонентах EditTexts класса Activity.

Листинг 10.18. Операторы package, import и переменные экземпляра класса AddEditContact

```
 1 // AddEditContact.java
 2 // Класс Activity, применяемый для добавления новой записи
 3 // либо изменения существующей записи в адресной книге.
 4 package com.deitel.addressbook;
 5
 6 import android.app.Activity;
 7 import android.app.AlertDialog;
 8 import android.os.AsyncTask;
 9 import android.os.Bundle;
10 import android.view.View;
11 import android.view.View.OnClickListener;
12 import android.widget.Button;
13 import android.widget.EditText;
```

```
public class AddEditContact extends Activity
{
   private long rowID; // id изменяемого контакта

   // Компоненты EditTexts, применяемые для ввода сведений о контакте
   private EditText nameEditText;
   private EditText phoneEditText;
   private EditText emailEditText;
   private EditText streetEditText;
   private EditText cityEditText;

```

Переопределение метода onCreate класса Activity

Метод onCreate (см. листинг 10.19) инициализирует метод AddEditContact класса Activity. В строках 33–37 обеспечивается доступ к компонентам EditText класса Activity. Затем с помощью метода getIntent класса Activity обеспечивается доступ к объекту Intent, который запустил Activity, и вызывается метод getExtras класса Intent. Этот метод обеспечивает доступ к классу Bundle, включающему дополнения класса Intent. После запуска метода AddEditContact класса Activity класса AddressBook Activity будет невозможно добавление каких-либо дополнений в Intent, поскольку пользователь указывает новую информацию контакта. В этом случае метод getExtras возвращает значение null. Если метод возвращает Bundle (строка 42), это означает, что класс Activity был запущен из класса ViewContact Activity, а пользователь выбрал изменение существующего контакта. В строках 44–49 считываются дополнения из Bundle путем вызова методов getLong (строка 44) и getString, а предназначенные для изменения данные String отображаются в компонентах EditText. В строках 53–55 регистрируется слушатель для кнопки Save Contact Button класса Activity.

Листинг 10.19. Переопределение методов onCreate и onPause класса Activity

```
26  // вызывается при первом запуске Activity
27  @Override
28  public void onCreate(Bundle savedInstanceState)
29  {
30     super.onCreate(savedInstanceState); // вызова onCreate суперкласса
31     setContentView(R.layout.add_contact); // «раздувание» UI
32
33     nameEditText = (EditText) findViewById(R.id.nameEditText);
34     emailEditText = (EditText) findViewById(R.id.emailEditText);
35     phoneEditText = (EditText) findViewById(R.id.phoneEditText);
36     streetEditText = (EditText) findViewById(R.id.streetEditText);
37     cityEditText = (EditText) findViewById(R.id.cityEditText);
38
39     Bundle extras = getIntent().getExtras(); // получение дополнений
40
41     // при наличии дополнений заполнение данными EditTexts
42     if (extras != null)
43     {
```

продолжение ⇗

Листинг 10.19 *(продолжение)*

```
         rowID = extras.getLong("row_id");
         nameEditText.setText(extras.getString("name"));
         emailEditText.setText(extras.getString("email"));
         phoneEditText.setText(extras.getString("phone"));
         streetEditText.setText(extras.getString("street"));
         cityEditText.setText(extras.getString("city"));
      }  // конец блока if

      // настройка слушателя событий для кнопки Save Contact Button
      Button saveContactButton =
         (Button) findViewById(R.id.saveContactButton);
      saveContactButton.setOnClickListener(saveContactButtonClicked);
   }  // конец описания метода onCreate

```

Объект OnClickListener, обрабатывающий события кнопки Save Contact Button

Как только пользователь касается кнопки Save Contact Button объекта AddEditContact класса Activity, вызывается метод saveContactButtonClicked класса OnClickListener (см. листинг 10.20). Чтобы сохранить контакт, пользователь должен указать, как минимум, название контакта. Метод onClick проверяет, чтобы длина имени контакта превышала 0 символов (строка 64), и если это так, создается и вызывается класс AsyncTask для выполнения операции сохранения. Метод doInBackground (строки 69–74) вызывает метод saveContact (см. листинг 10.21), выполняющий сохранение контакта в базе данных. Метод onPostExecute (строки 76–80) вызывает метод finish, который прекращает выполнение Activity и осуществляет возврат в запускающий класс Activity (AddressBook либо ViewContact). Если значение nameEditText пусто, в строках 89–96 отображается диалоговое окно AlertDialog, в котором отображается сообщение пользователю о необходимости указания имени контакта для его сохранения.

Листинг 10.20. Объект OnClickListener класса doneButtonClicked, обрабатывающий события doneButton

```
58    // отвечает на события, сгенерированные после щелчка пользователя
      // на кнопке Done Button
59    OnClickListener saveContactButtonClicked = new OnClickListener()
60    {
61       @Override
62       public void onClick(View v)
63       {
64          if (nameEditText.getText().length() != 0)
65          {
66             AsyncTask<Object, Object, Object> saveContactTask =
67                new AsyncTask<Object, Object, Object>()
68             {
69                @Override
70                protected Object doInBackground(Object... params)
71                {
```

```
                  saveContact(); // сохранение контакта в базе данных
                  return null;
               }  // конец определения метода doInBackground

               @Override
               protected void onPostExecute(Object result)
               {
                  finish(); // возврат к предыдущему Activity
               }  // конец описания метода onPostExecute
            };  // конец описания класса AsyncTask

         // сохранение контакта в базе данных с помощью отдельного потока
         saveContactTask.execute((Object[]) null);
      }  // конец блока if
      else
      {
         // создание нового AlertDialog Builder
         AlertDialog.Builder builder =
            new AlertDialog.Builder(AddEditContact.this);

         // настройка заголовка и сообщения диалогового окна и отображение
         // кнопки, предназначенной для скрытия диалогового окна
         builder.setTitle(R.string.errorTitle);
         builder.setMessage(R.string.errorMessage);
         builder.setPositiveButton(R.string.errorButton, null);
         builder.show(); // display the Dialog
      }  // конец else
   }  // конец описания метода onClick
};// конец описания OnClickListener saveContactButtonClicked
```

Метод saveContact

Метод saveContact (см. листинг 10.21) сохраняет информацию в компонентах EditText, относящихся к данному классу Activity. Сначала в строке 105 создается объект DatabaseConnectort, затем проверяется, имеет ли какие-либо расширения объект Intent, который запустил данный класс Activity. Если расширения отсутствуют, значит, мы имеем дело с новым контактом, поэтому в строках 110–115 получаем данные типа String из компонентов EditText класса Activity, которые затем передаются методу insertContact объекта DatabaseConnector для создания новых контактов. Если существуют дополнения для объекта Intent, который был запущен из текущего класса Activity, обновляется существующий контакт. В этом случае получаем данные типа String из компонентов EditText класса Activity, которые передаются методу updateContact класса DatabaseConnector. При этом с помощью идентификатора rowID указывается обновляемая запись. Методы insertContact и updateContact класса DatabaseConnector выполняют операции по открытию и закрытию базы данных.

Листинг 10.21. Метод saveContact класса AddEditContact

```
101   // сохраняет информацию о контакте в базе данных
102   private void saveContact()
```

продолжение ⇗

Листинг 10.21 *(продолжение)*

```
{
   // получение класса DatabaseConnector для взаимодействия
   // с базой данных SQLite
   DatabaseConnector databaseConnector = new DatabaseConnector(this);

   if (getIntent().getExtras() == null)
   {
      // включение в базу данных сведений о контакте
      databaseConnector.insertContact(
         nameEditText.getText().toString(),
         emailEditText.getText().toString(),
         phoneEditText.getText().toString(),
         streetEditText.getText().toString(),
         cityEditText.getText().toString());
   }  // конец блока if
   else
   {
      databaseConnector.updateContact(rowID,
         nameEditText.getText().toString(),
         emailEditText.getText().toString(),
         phoneEditText.getText().toString(),
         streetEditText.getText().toString(),
         cityEditText.getText().toString());
   } // конец блока else
}  // конец определения класса saveContact
} // конец определения класса AddEditContact
```

10.5.4. Класс утилиты DatabaseConnector

С помощью класса утилиты DatabaseConnector (см. листинги 10.22–10.25) выполняется управление взаимодействием приложения с базой данных SQLite. При этом создается база данных UserContacts с единственной таблицей именованных контактов, записи которой могут обрабатываться.

Операторы package, import и поля

В листинге 10.22 приведены операторы package, import и поля класса DatabaseConnector. Операторы import для новых классов и интерфейсов, рассмотренных в разделе 10.3, выделены другим цветом. Строковая константа DATABASE_NAME (строка 16) определяет имя базы данных, которая будет создана либо открыта. *Имена баз данных должны быть уникальными внутри одного приложения. Если же базы данных используются для различных приложений, уникальность не обязательна.* Объект SQLiteDatabase (строка 17) поддерживает доступ в режиме чтения/записи к базе данных SQLite. Класс DatabaseOpenHelper (строка 18) является частным вложенным классом, расширяющим абстрактный класс SQLiteOpenHelper. Последний класс применяется для управления, создания, открытия и обновления баз данных (возможно с одновременным изменением структуры базы данных). Код объекта SQLOpenHelper приведен в листинге 10.25.

Листинг 10.22. Операторы package, import и переменные экземпляра класса утилиты DatabaseConnector

```
// DatabaseConnector.java
// поддерживает простое подключение и создание базы данных UserContacts
package com.deitel.addressbook;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;

public class DatabaseConnector
{
   // имя базы данных
   private static final String DATABASE_NAME = "UserContacts";
   private SQLiteDatabase database; // объект базы данных
   private DatabaseOpenHelper databaseOpenHelper; // помощник
                                                  // базы данных

```

Конструктор и методы open и close класса DatabaseConnector

Конструктор класса DatabaseConnection (см. листинг 10.23, строки 21–26) создает новый объект класса DatabaseOpenHelper (см. листинг 10.25), который будет использоваться для открытия или создания базы данных. Код конструктора класса DatabaseOpenHelper приведен в листинге 10.25. Метод open (строки 29–33) пытается установить подключение к базе данных и генерирует исключение SQLException, если попытка подключения завершилась неудачей. Метод getWritableDatabase (строка 32), унаследованный от SQLiteOpenHelper, возвращает объект SQLiteDatabase. Если база данных не была создана, этот метод создаст базу данных. Если же база данных уже создана, метод открывает ее. Как только база данных успешно открыта, она *кэшируется* операционной системой для увеличения производительности будущих взаимодействий с базой данных. Метод close (строки 36–40) закрывает подключение к базе данных путем вызова метода close класса SQLiteOpenHelper.

Листинг 10.23. Конструктор, методы open и close

```
20   // общедоступный конструктор для DatabaseConnector
21   public DatabaseConnector(Context context)
22   {
23      // создание нового объекта DatabaseOpenHelper
24      databaseOpenHelper =
25         new DatabaseOpenHelper(context, DATABASE_NAME, null, 1);
26   } // конец определения конструктора DatabaseConnector
27
28   // открытие подключения к базе данных
29   public void open() throws SQLException
```

продолжение ☞

Листинг 10.23 *(продолжение)*

```
{
   // создание или открытие базы данных для чтения/записи
   database = databaseOpenHelper.getWritableDatabase();
}  // конец описания метода open

// закрытие подключения к базе данных
public void close()
{
   if (database != null)
   database.close(); // закрытие подключения к базе данных
}  // конец описания метода close

```

Методы insertContact, updateContact, getAllContacts, getOneContact и deleteContact

Метод insertContact (см. листинг 10.24, строки 43–56) вставляет новый контакт с относящейся к нему информацией в базу данных. Сначала часть информации контакта сохраняется в новом объекте ContentValues (строки 46–51), который поддерживает отображение ключа (пар «ключ–значение»). В качестве ключей используются имена столбцов базы данных. В строках 53–55 открывается база данных, вставляется новый контакт, а затем база данных закрывается. Метод insert класса SQLiteDatabase (строка 54) включает значения из данных переменных ContentValues в таблицу, определенную в качестве первого аргумента, — в данном случае используется таблица «контактов». Второй параметр этого метода, который не используется в настоящем приложении, — это именованный столбец nullColumnHack. Этот параметр требуется, поскольку *SQLite не поддерживает вставку абсолютно пустой строки в таблицу* — эквивалент передачи пустого объекта ContentValues методу insert. Вместо того чтобы выполнять недопустимую операцию передачи пустого объекта ContentValues методу, параметр nullColumnHack применяется для идентификации столбца, который принимает значения NULL.

Листинг 10.24. Методы insertContact, updateContact, getAllContacts, getOneContact и deleteContact

```
42  // вставка нового контакта в базу данных
43  public void insertContact(String name, String email, String phone,
44  String state, String city)
45  {
46     ContentValues newContact = new ContentValues();
47     newContact.put("name", name);
48     newContact.put("email", email);
49     newContact.put("phone", phone);
50     newContact.put("street", state);
51     newContact.put("city", city);
52
53     open(); // открыть базу данных
54     database.insert("contacts", null, newContact);
55     close(); // закрыть базу данных
56  }  // конец описания метода insertContact
```

```
57
58   // вставка нового контакта в базу данных
59   public void updateContact(long id, String name, String email,
60   String phone, String state, String city)
61   {
62      ContentValues editContact = new ContentValues();
63      editContact.put("name", name);
64      editContact.put("email", email);
65      editContact.put("phone", phone);
66      editContact.put("street", state);
67      editContact.put("city", city);
68
69      open(); // открыть базу данных
70      database.update("contacts", editContact, "_id=" + id, null);
71      close(); // закрыть базу данных
72   }  // конец описания метода updateContact
73
74   // возвращение в базу данных объекта Cursor вместе со всей
     // информацией о контакте
75   public Cursor getAllContacts()
76   {
77      return database.query("contacts", new String[] {"_id", "name"},
78         null, null, null, null, "name");
79     }   // конец описания метода getAllContacts
80
81   // получение объекта Cursor, включающего всю информаию о контакте
82   // с указанным идентификатором
83    public Cursor getOneContact(long id)
84   {
85      return database.query(
86         "contacts", null, "_id='" + id, null, null, null, null);
87   }  // конец описания метода getOnContact
88
89   // удаление контакта, указанного именем в формате String
90   public void deleteContact(long id)
91   {
92      open(); // открыть базу данных
93      database.delete("contacts", "_id=" + id, null);
94      close(); // закрыть базу данных
95   }  // конец описания метода deleteContact
96
```

Метод updateContact (строки 59–72) подобен методу insertContact за исключением того, что он вызывает метод update класса SQLiteDatabase (строка 70) для обновления существующего контакта. Третий аргумент метода update представляет конструкцию SQL WHERE (без ключевого слова WHERE), которая определяет обновляемую запись. В данном случае применяется идентификатор ID строки обновляемой записи, с помощью которого обновляется выбранный контакт.

Метод getAllContacts (строки 75–79) использует метод query класса SqLiteDatabase (строки 77–78) для выборки объекта Cursor, обеспечивающего доступ к идентификаторам ID и именам всех контактов в базе данных. Ниже перечислены аргументы этого метода:

- Имя таблицы, по отношению к которой создается запрос.
- Массив типа String для имен возвращаемых столбцов (здесь находятся _id и названия столбцов). Если в качестве этого параметра выбирается null, возвращаются все столбцы таблицы. Не следует использовать этот параметр, поскольку при этом расходуется много памяти, вычислительной мощности процессора и ускоренно разряжается батарея. Выбирайте только те данные, которые требуются.
- Конструкция SQL WHERE (без ключевого слова WHERE) либо значение null, возвращающее все строки.
- Массив аргументов типа String, которые подставляются в конструкцию WHERE. При этом символ ? используется в качестве символа подстановки для значения аргумента. На отсутствие аргументов в конструкции WHERE указывает значение null.
- Конструкция SQL GROUP BY (без ключевых слов GROUP BY) либо null, если не нужно группировать результаты.
- Конструкция SQL HAVING (без ключевого слова HAVING), с помощью которой определяются группы из конструкции GROUP BY, которые включаются в результаты выборки. Если вместо конструкции GROUP BY отображается null, следует указать null.
- Конструкция SQL ORDER BY (без ключевых слов ORDER BY) применяется для определения порядка следования результатов. Если использовано значение null, порядок следования результатов не указывается.

Объект Cursor, возвращаемый методом query, содержит все строки таблицы, которые соответствуют аргументам метода,— так называется *набор результатов*. Объект Cursor находится перед первой строкой набора результатов, причем различные методы move из Cursor могут применяться для перемещения Cursor среди набора результатов для выполнения обработки.

Метод getOneContact (строки 83–87) также использует метод query класса SqLiteDatabase для выполнения запроса к базе данных. В данном случае выбираются все столбцы из базы данных для контакта с указанным идентификатором ID.

Метод deleteContact (строки 90–95) использует метод delete класса SqLiteDatabase (строка 93) для удаления контакта из базы данных. В данном случае выбираются все столбцы базы данных, соответствующие контакту с указанным идентификатором ID. Три аргумента метода — это таблица базы данных, из которой удаляется запись, конструкция WHERE (без ключевого слова WHERE) и, в случае если конструкция WHERE включает аргументы, массив String значений, подставляемых в конструкцию WHERE (в нашем случае используется значение null).

Частный вложенный класс DatabaseOpenHelper, расширяющий класс LiteOpenHelper

Частный вложенный класс DatabaseOpenHelper (см. листинг 10.25) расширяет абстрактный класс SQLiteOpenHelper, который облегчает приложениям создание баз данных и управление изменениями версий. Конструктор (строки 100–104) просто вызывает конструктор суперкласса, который требует следующих четырех аргументов:

- аргумент Context, определяющий среду, в которой создается или открывается база данных;

- имя базы данных — здесь можно указать null, если используется база данных, находящаяся в оперативной памяти;
- используемый объект CursorFactory — если указан null, это означает, что используется заданный по умолчанию объект SQLite CursorFactory (для большинства приложений);
- номер версии базы данных (начиная с 1).

Листинг 10.25. Класс DatabaseOpenHelper класса SQLiteOpenHelper

```
 97 private class DatabaseOpenHelper extends SQLiteOpenHelper
 98 {
 99    // общедоступный конструктор
100    public DatabaseOpenHelper(Context context, String name,
101       CursorFactory factory, int version)
102    {
103       super(context, name, factory, version);
104    }  // конец определения конструктора DatabaseOpenHelper
105
106    // создание таблицы контактов в созданной базе данных
107    @Override
108    public void onCreate(SQLiteDatabase db)
109    {
110       // запрос на создание новой таблицы именованных контактов
111       String createQuery = "CREATE TABLE contacts" +
112          "(_id integer primary key autoincrement," +
113          "name TEXT, email TEXT, phone TEXT," +
114          "street TEXT, city TEXT);";
115
116       db.execSQL(createQuery); // выполнение запроса
117    }  // конец описания метода onCreate
118
119    @Override
120    public void onUpgrade(SQLiteDatabase db, int oldVersion,
121       int newVersion)
122    {
123    }  // конец описания метода onUpgrade
124  } // конец определения класса DatabaseOpenHelper
125 } // конец определения класса DatabaseConnector
```

Нужно переопределить абстрактные методы onCreate и onUpgrade. Если база данных на данный момент времени не существует, вызывается метод onCreate класса DatabaseOpenHelper, с помощью которого создается база данных. Если вы хотите использовать номер версии более новый, чем номер версии существующей базы данных, хранящейся на устройстве, вызывается метод onUpgrade класса DatabaseOpenHelper. Этот метод обновляет базу данных до новой версии (при этом могут добавляться таблицы или столбцы в существующую таблицу).

Метод onCreate (строки 107–117) определяет таблицу, создаваемую с помощью команды SQL CREATE TABLE, которая определена как String (строки 111–114). В нашем случае таблица контактов содержит поле целочисленного первичного ключа (_id), значение которого автоматически увеличивается на единицу, и текстовые поля для

всех остальных столбцов. В строке 116 вызывается метод execSQL класса SQLiteDatabase, с помощью которого выполняется команда CREATE TABLE. Поскольку нам не нужно обновлять базу данных, мы просто переопределяем метод onUpgrade пустой «заглушкой». Как и в Android 3.0, класс SQLiteOpenHelper также поддерживает метод onDowngrade, который используется для отката базы данных в случае, если текущий сохраненный номер версии больше, чем номер версии, указанный при вызове конструктора SQLiteOpenHelper. С помощью отката можно вернуть базу данных к предыдущей версии, когда в таблицах содержалось меньше столбцов, а количество самих таблиц было меньше. Это может потребоваться для устранения ошибок в приложении.

Для всех методов класса SQLiteDatabase, используемых в классе DatabaseConnector, существуют соответствующие методы, выполняющие те же самые операции, но при этом генерирующие исключения в случае ошибок. Это лучше, чем возврат значения -1 в случае ошибки (например, может использоваться метод insertOrThrow вместо метода insert). Эти методы взаимозаменяемы, и пользователь может сам выбрать методы обработки ошибок, возникающих при чтении и записи в базу данных.

10.6. Резюме

На протяжении этой главы мы создали приложение Address Book, с помощью которого пользователи могут добавлять, просматривать, изменять либо удалять информацию о контактах, которая хранится в базе данных SQLite. Был изучен каждый класс Activity приложения, описанный в файле манифеста приложения AndroidManifest.xml.

Мы определили пары атрибут–значение общих компонентов GUI в виде XML-ресурсов style. Затем применили стили ко всем компонентам, которые совместно используют эти значения с помощью атрибута компонента style. Мы добавили границы в компоненты TextView путем указания Drawable в качестве значения атрибута android:background компонента TextView, а также создали пользовательский Drawable с помощью XML-представления формы.

С помощью XML-ресурсов меню мы определили элементы MenuItems приложения, которые были программно «раздуты» с помощью MenuInflater класса Activity. С помощью стандартных пиктограмм Android улучшили отображение элементов меню.

Как известно, основная задача класса Activity заключается в отображении списков элементов с полосами прокрутки. В этой главе вы научились использовать расширенный класс ListActivity, с помощью которого создается класс Activity, отображающий компонент ListView в заданном по умолчанию макете. Этот класс использовался для отображения контактов, хранящихся в базе данных приложения. Вы узнали о том, что компонент ListView является подклассом класса AdapterView, обеспечивающего связывание компонента с источником данных. Также класс CursorAdapter использовался для отображения результатов запроса к базе данных в основном компоненте ListView класса Activity.

Явно определенные объекты Intents применялись для запуска новых деятельностей, выполняющих такие задачи, как добавление контакта, изменение либо удаление существующего контакта. Мы изучили, как завершать запущенную деятельность для возврата в предыдущее состояние с помощью метода finish класса Activity.

Мы использовали подкласс класса SQLiteOpenHelper в целях упрощения создания базы данных и получения объекта SQLiteDatabase, с помощью которого выполняется

манипулирование содержимым базы данных. Класс Cursor использовался для обработки результатов запроса. Подклассы класса AsyncTask применялись для выполнения задач базы данных за пределами потока GUI и возвращения результатов в поток GUI, благодаря чему можно использовать преимущества возможностей потоков Android без непосредственного создания и манипулирования потоками.

В главе 11 вашему вниманию будет представлено приложение Route Tracker, использующее технологию GPS для отслеживания местоположения пользователя и его отметки на слое карты города, нарисованной поверх слоя спутниковой карты. Это приложение использует компонент MapView для взаимодействия с веб-службами Google Maps и отображения карт. С помощью объекта Overlay отображается местоположение пользователя. Приложение также принимает данные GPS и информацию о направлении движения от служб геолокации и сенсоров Android.

11

Приложение Route Tracker

Google Maps API, GPS, LocationManager, MapActivity, MapView и Overlay

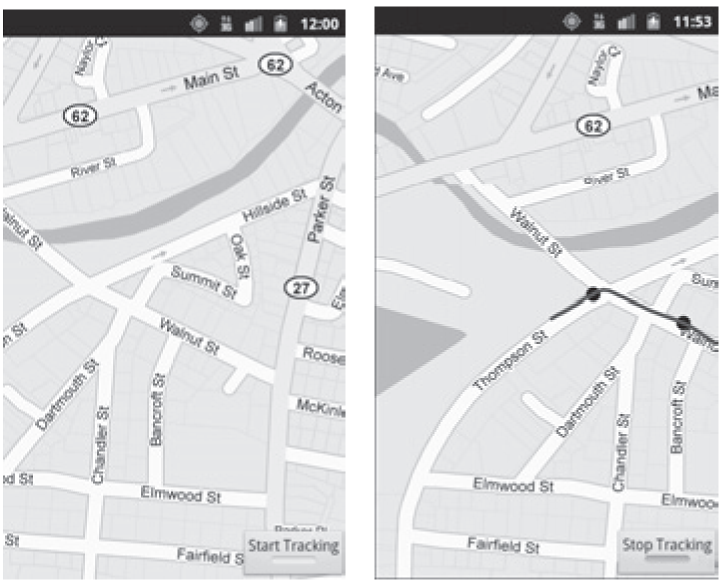
В этой главе...

- Тестирование приложения, использующего GPS-данные о местоположении, с помощью Android Emulator, а также использование перспективы Eclipse DDMS для отправки образцов данных GPS эмулятору.
- Использование внешнего фреймворка Maps API, а также классов MapActivity и MapView для отображения карт Google Maps™, сгенерированных веб-службами Google.
- Получение ключа Google Maps™ API, уникального для компьютера разработчика.
- Использование служб геолокации и класса LocationManager для получения сведений о местонахождении и направлении движения устройства.
- Отображение маршрута пользователя с помощью класса Overlay, формирующего отображение на компоненте MapView, и данных локации GPS, принимаемых в форме объектов Location.
- Ориентирование карты в соответствии с направлением движения пользователя.
- Использование PowerManager для «пробуждения» устройства

11.1. Введение

Во время путешествия пользователя вместе с устройством Android приложение Route Tracker отслеживает *местоположение* и *направление движения*. Маршрут движения отображается на карте. Чтобы начать отслеживание движения по маршруту, коснитесь переключателя Start Tracking ToggleButton (Начать отслеживание), как показано на рис. 11.1, *а*. (*Переключатель* — это кнопка, имеющая два положения (включено

и отключено).) После этого изменяется надпись переключателя ToggleButton на Stop Tracking (Прекратить отслеживание) и отображается зеленая панель — индикатор отслеживания приложением маршрута движения пользователя. По мере движения по маршруту происходит смещение карты, причем текущее местоположение остается в центре экрана (рис. 11.1, *б*). Маршрут — это красная линия с черными точками, которая появляется после каждых 10 точек данных GPS, принятых приложением (см. рис. 11.1, *б*). Если это приложение установлено на устройстве Android, то карта ориентируется таким образом, что линия отслеживания маршрута указывает направление перемещения (*движение по маршруту*), которое, в свою очередь, направлено вверх экрана устройства. Иллюстрации, приведенные в главе, были созданы с помощью приложения, установленного на эмуляторе Android, который не может имитировать данные о положении устройства относительно сторон света. В меню приложения можно выбрать параметр Map (Карта) или Satellite (Спутник), как показано на рис. 11.2, *а*. В результате изменяется выбранный стиль карты. После выбора параметра Map отображается карта в *режиме просмотра улиц* Google™ Maps (режим просмотра приложения, *заданный по умолчанию*). В результате выбора параметра Satellite (Спутник) отображается *полученная со спутника фотография местности*, на которой находится пользователь (см. рис. 11.2, *б*). Для завершения отслеживания текущего маршрута коснитесь переключателя Stop Tracking ToggleButton. После этого приложение выведет диалоговое окно, в котором поверх маршрута (рис. 11.3) отобразится пройденное расстояние (в километрах и милях), а также средняя скорость перемещения (в км/ч и милях/ч).



а *б*

Рис. 11.1. Окно приложения Route Tracker до и после выбора переключателя Start Tracker: *а* — пользователь выбрал переключатель Start Tracking для начала отслеживания маршрута; *б* — пользователь выбрал переключатель Stop Tracking для завершения отслеживания маршрута



а *б*

Рис. 11.2. В этом меню пользователь может выбрать режим просмотра карты или спутникового изображения; после выбора параметра Satellite приложение отображает снимок местности из космоса: *а* — в меню открывается доступ к параметрам Map и Satellite; *б* — режим просмотра Satellite выбирается после касания пункта меню Satellite

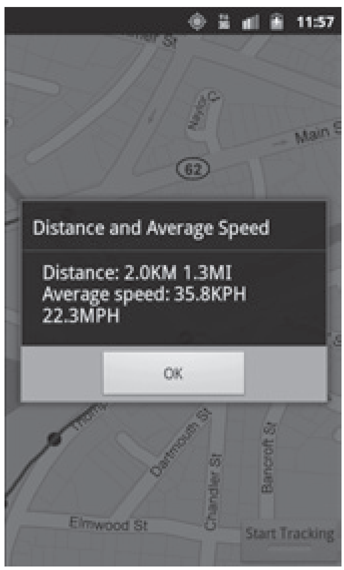


Рис. 11.3. После выбора пользователем параметра Stop Tracking отображается статистика

11.2. Тестирование приложения Route Tracker

Импорт приложения

Откройте среду Eclipse и импортируйте проект Route Tracker app. Выполните следующие действия:

1. Выполните команды File ▸ Import... (Файл ▸ Импорт...) для открытия диалогового окна Import (Импорт).
2. В диалоговом окне Import раскройте узел General (Общий) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область). Затем щелкните на кнопке Next > (Далее >).
3. Справа от текстового поля Select root directory (Выбрать корневой каталог) щелкните на кнопке Browse... (Просмотр...). Затем найдите и выделите папку Route Tracker.
4. Щелкните на кнопке Finish (Готово) для выполнения импорта проекта.

Получение ключа Google Maps API

Чтобы запустить на выполнение приложение Route Tracker либо создать собственное приложение, использующее возможности Google Maps API, получите уникальный *ключ API* от Google. Прежде чем выдать вам ключ, Google потребует «отпечатки пальцев», которые уникальным образом идентифицируют компьютер разработчика. Как вы помните из раздела 2.7, перед установкой приложений на устройстве следует их подписать с помощью цифрового сертификата. При создании и тестировании приложений модуль ADT Plugin выполняет эту операцию автоматически путем создания *сертификата отладки*, с помощью которого подписываются приложения. «Отпечатки пальцев», требуемые Google (известные под названием *MD5 Fingerprint*), могут генерироваться с помощью сертификата отладки. Ключ API, полученный с помощью «отпечатков пальцев», может применяться исключительно для отладки и тестирования приложений. Дополнительные сведения о технологиях *MD5 encryption* и *MD5 fingerprints* можно найти на веб-сайтах:

- en.wikipedia.org/wiki/Md5;
- en.wikipedia.org/wiki/Public_key_fingerprint.

Придерживайтесь инструкций, изложенных на веб-сайте code.google.com/android/add-ons/google-apis/mapkey.html в разделе «Getting the MD5 Fingerprint of the SDK Debug Certificate». Затем воспользуйтесь значением fingerprint, которое можно получить на веб-сайте code.google.com/android/maps-api-signup.html для генерирования вашего уникального ключа Google Maps API. Если вы планируете распространять разработанное приложение, следуйте инструкциям, приведенным в разделе «Getting the MD5 Fingerprint of Your Signing Certificate» для получения отдельного ключа Google Maps API. Этот раздел находится на первом веб-сайте из приведенного выше списка.

ПРИМЕЧАНИЕ

Чтобы протестировать это приложение, вместо значения String ресурса google_maps_api_key в файле strings.xml подставьте ваш собственный ключ Google Maps API. Если этого не сделать, приложение будет выполняться без отображения карт и спутниковых фотографий, известных как *карточные плитки*.

Выполнение и тестирование приложения на устройстве Android

Чтобы протестировать приложение на реальном устройстве Android, подключитесь к Интернету (для получения изображений карты). Правильно настройте устройство, применяемое для тестирования и отладки приложений (в соответствии с рекомендациями из раздела «Подготовительные действия» вводной главы книги), и подключите его к компьютеру. Щелкните правой кнопкой мыши на проекте приложения в окне Package Explorer, отображаемого в среде Eclipse, затем в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android). Если отобразилось диалоговое окно Android Device Chooser (Выбор устройства Android), выберите устройства и щелкните на кнопке OK для установки и выполнения приложения на устройстве.

Для приема сигналов спутников системы GPS ваше устройство должно находиться в условиях прямой видимости спутников. Для этого лучше всего выйти на улицу и подождать несколько минут, необходимых для поиска и приема сигнала GPS устройством. После запуска приложения Route Tracker на устройстве выйдите из помещения на улицу. После приема устройством сигнала GPS на экране появится кратковременное сообщение Toast, свидетельствующее об этом событии. Теперь коснитесь переключателя Start Tracking и несколько минут походите туда-сюда.

По мере вашего перемещения на экране будет отображаться маршрут в виде красной линии. Если устройство снабжено компасом, приложение ориентирует карту в соответствии с направлением, в котором ориентировано устройство. Если в устройстве отсутствует азимутальный сенсор, настройки ориентации карты не происходит. Откройте меню приложения и выберите пункт Satellite для отображения спутниковой фотографии вместо стандартного изображения карты. Чтобы вернуться обратно к режиму просмотра улиц, воспользуйтесь пунктом меню Map. После завершения прохождения маршрута выберите переключатель Stop Tracking. На экране появится диалоговое окно AlertDialog, в котором отображается пройденное расстояние и средняя скорость прохождения маршрута. Коснитесь кнопки OK для закрытия этого окна и возврата к листу карты. Чтобы просмотреть только что пройденный маршрут, воспользуйтесь панорамированием (выполняется путем перетаскивания пальцем области карты); масштаб просмотра карты меняется с помощью жестов сведения/разведения пальцев. В результате повторного выбора переключателя Start Tracking предыдущий маршрут удаляется и начинается отслеживание нового маршрута.

Выполнение приложения на устройстве AVD

Для выполнения этого приложения на виртуальном устройстве AVD сконфигурируйте виртуальное устройство AVD таким образом, чтобы использовать Google API совместно с версией вашей платформы Android. Выполните следующие действия:

1. Откройте окно Android SDK and AVD Manager.
2. Выберите одно из устройств Android AVD, которые были сконфигурированы в разделе «Подготовительные действия» вводной главы (в этой главе используется устройство NexusS), а затем щелкните на кнопке Edit... (Изменить...).
3. В окне Edit Android Virtual Device (AVD) (Изменить виртуальное устройство Android (AVD)) в раскрывающемся списке Target (Цель) выберите параметр Google APIs

(Google Inc.) — API Level #, а затем щелкните на кнопке Edit AVD (Изменить AVD). В данном случае значок # обозначает уровень (level) API, используемой для разработки. В результате виртуальное устройство AVD будет использовать Android API и Google API для выбранного уровня API (например, API уровня 10 соответствует Android 2.3.3). Если вы предпочитаете не изменять существующее устройство AVD, создайте отдельное устройство AVD, воспользовавшись методиками, описанными в разделе «Подготовительные действия» вводной главы.

4. В окне Android SDK and AVD Manager выберите устройство AVD и запустите его.

Щелкните правой кнопкой мыши на проекте приложения в окне Package Explorer среды Eclipse, в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android). В отобразившемся окне Android Device Chooser выберите устройство AVD, а затем щелкните на кнопке OK для установки приложения и выполнения его на этом AVD.

Отправка GPS-данных на устройство AVD с помощью GPX-файлов

Эмулятор Android позволяет отсылать GPS-данные устройству AVD, в результате чего возможно тестирование приложений, определяющих местоположение, без физического устройства Android. Для выполнения подобного тестирования воспользуйтесь файлом, содержащим GPS-данные в формате GPS Exchange Format. Подобные файлы обычно имеют расширение .gpx и называются GPX-файлами. Вместе с примерами книги пользователю предлагаются несколько GPX-файлов (в папке GPXfiles), которые можно загрузить и «проиграть» в перспективе DDMS из плагина ADT. В результате GPS-данные пересылаются выбранному устройству AVD. Файлы GPX записываются с помощью бесплатно распространяемого приложения GPSLogger, которое можно найти на Google Play: play.google.com/store/apps/details?id=com.mendhak.gpslogger.

Данные GPS, сохраненные в GPX-файлах, представляют описания кратких автомобильных поездок в штате Массачусетс. Инструмент GPSLogger сохраняет файлы в формате GPX версии 1.0, а эмулятор Android использует формат GPX версии 1.1. В Интернете можно найти множество инструментов, выполняющих преобразования между различными форматами данных GPS. Мы воспользовались инструментом, доступным на веб-сайте www.gpsbabel.org.

Этот инструмент обеспечивает открытие каждого файла и его сохранение в формате GPX 1.1.

Для отправки GPS-данных из файла GPX устройству AVD, выполните следующие действия:

1. Во время выполнения приложения на устройстве AVD в среде Eclipse выполните команды Window ▸ Open Perspective ▸ DDMS (Окно ▸ Открыть перспективу ▸ DDMS), чтобы переключиться на перспективу DDMS.
2. На вкладке Devices (Устройства) выберите ваше устройство AVD.
3. На вкладке Emulator Control (Управление эмулятором) выберите вкладку GPX.
4. Щелкните на кнопке Load GPX... (Загрузить GPX...), затем найдите и выберите один из GPX-файлов, находящихся в папке GPXFiles, которая, в свою очередь, находится в папке примеров книги, и щелкните на кнопке Open (Открыть).

5. В нижней половине вкладки GPX выберите только что открытый файл и щелкните на кнопке воспроизведения. В результате начнется пересылка GPS-данных файла выбранному устройству AVD.

В окне AVD выберите параметр Start Tracking и наблюдайте, как обновляется маршрут по мере получения приложением образца GPS-данных. После выбора параметра Stop Tracking приложение отобразит сообщение о длине пройденного маршрута и средней скорости его прохождения для данных, полученных приложением.

11.3. Обзор применяемых технологий

В этом разделе будут представлены новые технологии, используемые при разработке приложения Route Tracker (в порядке их применения в данной главе).

Новые функции, определенные в файле AndroidManifest.xml

Данное приложение использует несколько новых функций, описанных в файле манифеста приложения (рассмотрены в разделе 11.4):

- Для получения доступа к нестандартной библиотеке, которая не входит в число основных библиотек Android API (например, Google Maps API), укажите название библиотеки в манифесте приложения. При этом используется элемент uses-library, включенный в элемент application.
- Если требуется, чтобы приложение использовало большую часть экрана для отображения карт, можно скрыть строку заголовка. Для этого воспользуйтесь одной из стандартных *тем* Android, которая выбирается путем указания значения атрибута android:theme для элемента activity. Тема полностью изменяет внешний вид интерфейса пользователя приложения. Список предопределенных стилей и тем Android можно найти на веб-сайте developer.android.com/reference/android/R.style.html.

По умолчанию общедоступные службы Android недоступны из приложения. В число подобных служб входят службы, позволяющие изменять настройки электропитания, получать данные о местоположении, контролировать переход устройства в режим сна и ряд других настроек. Для получения доступа к этим службам запросите разрешение на использование подобной службы в файле манифеста. При этом используются элементы uses-permission, вложенные в корневой элемент manifest. Если пользователь собирается установить приложение, операционная система сообщает о необходимых для этого разрешениях, а также требует подтверждения предоставления подобных разрешений. Если пользователь не подтвердит разрешение, приложение не будет установлено. Полный список разрешений доступен на веб-сайте developer.android.com/reference/android/Manifest.permission.html.

Класс ToggleButton

Класс ToggleButton (пакет android.widget) поддерживает состояние on–off (включено-выключено). Изначально класс ToggleButton приложения отображает текст Start Tracking с находящейся ниже серой полосой, сигнализирующей о том, что для кнопки выбрано состояние off. Чтобы начать отслеживать маршрут, пользователю достаточно коснуться

этой кнопки. После этого надпись кнопки ToggleButton изменяется на Stop Tracking, под кнопкой отображается зеленая полоса, свидетельствующая о том, что кнопка находится в состоянии on (включено), а приложение начало отслеживать маршрут. Как только пользователь коснется этой кнопки снова, она переключится обратно в состояние off (снова отобразится текст Start Tracking), а приложение прекращает отслеживание маршрута и отображает в диалоговом окне результаты отслеживания. Класс ToggleButton является подклассом класса CompoundButton. События класса CompoundButton обрабатываются путем реализации интерфейса CompoundButton.OnCheckedChangeListener.

Классы MapActivity, MapView и Overlay

Пакет com.google.android.maps включает классы, используемые для взаимодействия с Google Maps API. Класс RouteTracker (раздел 11.5.1) является подклассом класса MapActivity — деятельности, управляющей компонентом MapView (раздел 11.5.2). Этот класс применяется для отображения карт, полученных с помощью Google Maps API. Компоненты MapView поддерживают жесты, предназначенные для масштабирования и панорамирования карты. Любые другие дополнительные функции должны добавляться программно. Для отображения данных с помощью MapView, например линии, с помощью которой представлен маршрут приложения, создается подкласс класса Overlay (раздел 11.5.3) и переопределяется его метод draw. С помощью объектов GeoPoints (разделы 11.5.1 и 11.5.3) GPS-данные транслируются в точки, которые могут использоваться для рецентрирования карты на основе местоположения пользователя и прокладки маршрута.

Данные о местоположении

Пакет android.location (раздел 11.5.1) включает классы и интерфейсы, предназначенные для получения и использования данных о местоположении. Класс LocationManager обеспечивает доступ к службам геолокации устройства. Эти службы зависят от используемого оборудования и могут применяться для периодического обновления сведений о местоположении устройства либо для запуска объекта Intent в случае, если пользователь направляется в определенный географический регион. В зависимости от используемого устройства поддерживаются несколько провайдеров геолокации. Класс LocationManager обеспечивает возможность выбора лучшего провайдера на основании требований приложения, определенных с помощью объекта Criteria. Настройки, которые могут быть указаны с помощью Criteria, — точность, использование батареи, отслеживание перемещения, скорость, высота и стоимость услуг провайдера. И поскольку вы располагаете поставщиком данных о местоположении, вы можете запрашивать у него обновленные данные о вашем местоположении, передавая их LocationListener. Обновленные сведения передаются слушателю в качестве объектов Location, которые представляют географическое местоположение устройства, — сведения о широте и долготе, время записи данных и (в зависимости от поставщика данных о местоположении) данные о высоте и скорости перемещения (у некоторых устройств просто отсутствуют соответствующие сенсоры). Чтобы определить наличие «захвата» сигнала GPS, то есть приема сигналов GPS с достаточно большого количества спутников, необходимого для точного отслеживания местоположения, мы реализуем интерфейс GpsStatus.Listener.

Классы PowerManager и WakeLock

Класс PowerManager (пакет android.os) обеспечивает приложению возможность управления питанием устройства Android. Приложение, изменяющее настройки питания, может оказать отрицательное влияние на время «жизни» батареи устройства при выполнении приложения, поэтому следует параллельно использовать класс PowerManager. Как только пользователь начинает отслеживать маршрут, он, разумеется, хочет, чтобы приложение фиксировало данные о местоположении, даже если отключен экран устройства. Мы используем класс PowerManager для перехода в режим WakeLock, в котором предотвращается переход устройства в режим сна, в результате чего приложение может продолжать принимать GPS-данные (раздел 11.5.1).

Программное определение размера экрана устройства

Класс Display (пакет android.view) обеспечивает доступ к размерам экрана устройства. Эти размеры используются (раздел 11.5.2) для облегчения масштабирования карты, в результате чего она будет при повороте устройства заполнять экран в соответствии с направлением движения пользователя.

11.4. Создание графического интерфейса пользователя и файлов ресурсов

В этом разделе будут созданы файлы ресурсов и файлы разметки графического интерфейса пользователя для приложения Route Tracker. Для экономии места на страницах книги мы не будем приводить код файла ресурсов strings.xml и файла разметки меню приложения. Содержимое этих файлов можно просмотреть, открыв их в проекте Eclipse.

11.4.1. Создание проекта

Начните с создания нового проекта Android под названием RouteTracker. В диалоговом окне New Android Project (Новый проект Android) укажите следующие значения, потом нажмите кнопку Finish (Готово):

- Build Target (Операционная система). Проверьте, чтобы были выбраны Google API для платформы 2.3.3 (либо более поздней). В результате модуль ADT Plugin включит в проект Android API и Google API для Android 2.3.3 (либо выбранной вами версии). Также будут включены Google API для Google Maps.
- Application name (Имя приложения): Route Tracker.
- Package name (Название пакета): com.deitel.routetracker.
- Create Activity (Создать деятельность): RouteTracker.
- Min SDK Version (Минимальная версия SDK): 8.

11.4.2. Файл AndroidManifest.xml

В листинге 11.1 представлен код файла AndroidManifest.xml для этого приложения. В этом листинге выделены некоторые новые свойства.

Листинг 11.1. Файл AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android=http://schemas.android.com/apk/res/android
3     package="com.deitel.routetracker" android:versionCode="1"
4     android:versionName="1.0">
5     <application android:icon="@drawable/icon"
6        android:label="@string/app_name" android:debuggable="true">
7        <uses-library android:name="com.google.android.maps" />
8        <activity android:name=".RouteTracker"
9           android:label="@string/app_name"
10          android:theme="@android:style/Theme.Black.NoTitleBar"
11          android:screenOrientation="portrait">
12          <intent-filter>
13             <action android:name="android.intent.action.MAIN" />
14             <category android:name="android.intent.category.LAUNCHER" />
15          </intent-filter>
16       </activity>
17
18    </application>
19    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="10"/>
20
21    <uses-permission android:name="android.permission.INTERNET" />
22    <uses-permission
23       android:name="android.permission.ACCESS_FINE_LOCATION" />
24    <uses-permission
25       android:name="android.permission.ACCESS_MOCK_LOCATION" />
26    <uses-permission android:name="android.permission.WAKE_LOCK" />
27 </manifest>
```

Использование внешней библиотеки

В строке 7 объявляется использование в приложении библиотеки Google Maps API с помощью элемента uses-library, вложенного в элемент application.

Скрытие заголовка приложения

В строке 10 с помощью атрибута android:theme элемента activity в качестве темы Activity выбирается Theme.Black.NoTitleBar — вариация стандартной темы Android, в которой просто скрыта строка заголовка Activity.

Запрос разрешений приложения

Элементы uses-permission в строках 21–26 указывают на то, что приложение будет корректно выполняться только в случае предоставления следующих разрешений:

- android.permission.INTERNET. Приложение нуждается в доступе к Интернету для загрузки карт и спутниковых фотографий.
- android.permission.ACCESS_FINE_LOCATION. Приложение требует точных данных о местоположении для отображения маршрута пользователя на карте.

- android.permission.ACCESS_MOCK_LOCATION. Это приложение должно принимать имитированные данные для выполнения тестирования (как показано в разделе 11.2). Это разрешение требуется лишь на этапе разработки приложения.
- android.permission.WAKE_LOCK. Приложение требует доступа к классу PowerManager для предотвращения перехода в режим сна во время прокладывания маршрута.

Дополнительные сведения о разрешениях Android и модели безопасности можно найти на веб-сайте developer.android.com/guide/topics/security/security.html.

11.4.3. Разметка приложения Route Tracker: файл main.xml

XML-разметка приложения Route Tracker (листинг 11.2) включает класс FrameLayout (пакет android.widget) со *стеками*, заданными по умолчанию (*слоями*). Последние добавленные компоненты слоя находятся сверху. Компоненты позиционируются в верхнем левом углу FrameLayout, если только для позиционирования не используется свойство gravity. В данной разметке предусматривается расположение переключателя ToggleButton в правом нижнем углу. Мы программным образом добавили в эту разметку объект класса BearingFrameLayout, содержащего компонент MapView, используемый для отображения маршрута. Атрибуты android:textOn и android:textOff (строки 9–10) класса ToggleButton позволяют определить текст, отображаемый на кнопке для состояний «включено» и «выключено» соответственно.

Листинг 11.2. Разметка для подкласса RouteTracker класса MapActivity

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <FrameLayout xmlns:android=http://schemas.android.com/apk/res/android
 3    android:id="@+id/mainLayout"
 4    android:layout_width="match_parent"
 5    android:layout_height="match_parent">
 6    <ToggleButton android:id="@+id/trackingToggleButton"
 7       android:layout_width="wrap_content"
 8       android:layout_height="wrap_content"
 9       android:textOn="@string/button_stop_tracking"
10       android:textOff="@string/button_start_tracking"
11       android:layout_gravity="bottom|right"></ToggleButton>
12    </FrameLayout>
```

11.5. Создание приложения

Это приложение состоит из классов RouteTracker (подкласс класса MapActivity; листинги 11.3–11.11), BearingFrameLayout (листинги 11.12–11.16) RouteOverlay (листинги 11.17–11.20). Как и в предыдущих приложениях, основной класс Activity этого приложения — RouteTracker — создается при создании проекта, но вы можете изменить его суперкласс на MapActivity непосредственно в исходном коде. Другие классы нужно добавить в папку src/com.deitel.routetracker проекта.

11.5.1. Подкласс RouteTracker класса MapActivity

Класс RouteTracker (см. листинги 11.3–11.11) — это класс Activity приложения. Как отмечалось ранее, этот класс расширяет класс MapActivity, поскольку основное назначение данного класса Activity заключается в отображении компонента MapView, который, в свою очередь, отображает карту Google Map. Как и класс ListActivity, класс MapActivity обеспечивает поддержку «жизненного цикла» для компонентов View. В настоящее время поддерживается лишь один класс MapActivity для одного процесса.

Операторы package, import и поля класса RouteTracker

В листинге 11.3 показаны операторы package, import, а также поля класса RouteTracker. Мы выделили операторы import для новых классов и интерфейсов, которые были рассмотрены в разделе 11.3, а также будут рассмотрены в разделе 11.5.1. Далее будут рассмотрены переменные и константы экземпляра этого класса.

Листинг 11.3. Операторы package, import и переменные экземпляра подкласса RouteTracker класса MapActivity

```
// RouteTracker.java
// Основной класс MapActivity для приложения RouteTracker.
package com.deitel.routetracker;

import android.app.AlertDialog;
import android.content.Context;
import android.location.Criteria;
import android.location.GpsStatus;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.PowerManager;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.FrameLayout;
import android.widget.Toast;
import android.widget.ToggleButton;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;

public class RouteTracker extends MapActivity
{
   private LocationManager locationManager; // данные о локации
   private MapView mapView; // карта Google
```

продолжение ⇏

Листинг 11.3 *(продолжение)*

```
   private MapController mapController; // панорамирование/
                                        // масштабирование карты
   private Location previousLocation; // предыдущая локация
   private RouteOverlay routeOverlay; // отображение маршрута на карте
   private long distanceTraveled; // общее расстояние
   private BearingFrameLayout bearingFrameLayout; // вращение MapView
   private boolean tracking; // режим отслеживания?
   private long startTime; // время отслеживания (в миллисекундах)
   private PowerManager.WakeLock wakeLock; // блокирование перехода
                                           // в режим сна
   private boolean gpsFix; // «захват» GPS?

   private static final double MILLISECONDS_PER_HOUR = 1000 * 60 * 60;
   private static final double MILES_PER_KILOMETER = 0.621371192;
   private static final int MAP_ZOOM = 18; // Google Maps supports 1-21

```

Переопределение метода onCreate класса Activity

В листинге 11.4 переопределяется метод onCreate класса Activity. В строках 55–56 переменной экземпляра класса bearingFrameLayout присваивается новый объект класса BearingFrameLayout (раздел 11.5.2), который создает компонент MapView и вращает его в соответствии с направлением движения пользователя. В результате карта будет сориентирована в направлении движения пользователя (эта функция не поддерживается в эмуляторе Android). В строке 64 компонент MapView получается из BearingFrameLayout и присваивается переменной экземпляру mapView. В строке 65 с помощью метода getController открывается доступ к объекту MapController из mapView. С помощью MapController выполняется программное масштабирование карты, а также изменяется географическая локация, отображаемая в центре MapView. В строке 66 используется метод setZoom класса MapController для настройки уровня масштаба карты (то есть уровень деталировки). Уровни масштабирования находятся в диапазоне от 1 (минимальный масштаб) до 21 (максимальный масштаб). По мере увеличения масштаба просмотра карты каждый следующий уровень приводит к уменьшению в два раза размеров отображаемой местности. Иногда при предельном увеличении масштаба на экране не отображается ни карта, ни спутниковая фотография по причине отсутствия подобных карт либо фотографий у Google.

Листинг 11.4. Переопределение метода onCreate класса Activity

```
47 // Вызывается при первом создании деятельности
48 @Override
49 public void onCreate(Bundle savedInstanceState)
50 {
51    super.onCreate(savedInstanceState);
52    setContentView(R.layout.main);
53
54    // создание нового MapView с помощью ключа Google Maps API
55    bearingFrameLayout = new BearingFrameLayout(this,
56       getResources().getString(R.string.google_maps_api_key));
```

```
57
58       // добавление bearingFrameLayout в основную разметку mainLayout
59       FrameLayout mainLayout =
60          (FrameLayout) findViewById(R.id.mainLayout);
61       mainLayout.addView(bearingFrameLayout, 0);
62
63       // получение MapView и MapController
64       mapView = bearingFrameLayout.getMapview();
65       mapController = mapView.getController(); // получение MapController
66       mapController.setZoom(MAP_ZOOM); // увеличение масштаба карты
67
68      // создание слоя карты
69      routeOverlay = new RouteOverlay();
70
71      // добавление слоя RouteOverlay
72      mapView.getOverlays().add(routeOverlay);
73
74      distanceTraveled = 0; // инициализация distanceTraveled значением 0
75
76      // регистрация слушателя для trackingToggleButton
77      ToggleButton trackingToggleButton =
78         (ToggleButton) findViewById(R.id.trackingToggleButton);
79      trackingToggleButton.setOnCheckedChangeListener(
80      trackingToggleButtonListener);
81   }  // конец определения метода onCreate
82
```

В строке 69 переменной экземпляра класса routeOverlay присваивается новый объект подкласса Overlay класса RouteOverlay (раздел 11.5.3), который используется для отображения маршрута пользователя с помощью MapView. Потом в строке 72 обеспечивается доступ к коллекции объектов Overlay из компонента mapView и добавляется объект routeOverlay в эту коллекцию. Для каждого объекта Overlay, отображаемого на карте, сохраняется одна и та же ориентация и масштаб.

В строке 74 переменной экземпляра класса distanceTraveled присваивается значение 0. В процессе отслеживания маршрута приложение обновляет значение переменой distanceTraveled после приема каждой новой точки данных GPS. В строках 77–80 обеспечивается доступ к компоненту trackingToggleButton и регистрируется слушатель trackingToggleButtonListener (см. листинг 11.11) в качестве OnCheckedChangeListener этого компонента.

Переопределение методов onStart и onStop класса Activity

В листинге 11.5 переопределяются методы onStart и onStop класса Activity. Выполнение метода onStart (строки 84–121) начинается с конфигурирования объекта Criteria, который представляет запрошенные функции и настройки для *провайдера локации*. В строках 91–95 вызываются методы класса Criteria, с помощью которых определяются следующие настройки:

- setAccuracy. Константа Criteria.ACCURACY_FINE указывает, требует ли приложение точных GPS-данных, позволяющих отслеживать маршрут пользователя

с максимально возможной точностью. Точные данные GPS требуют большой вычислительной мощности и приводят к ускоренному разряду батареи. И если приложение не нуждается в подобной точности, выберите константу Criteria.ACCURACY_COARSE. Как и в Android 2.3, доступны следующие три уровня точности: Criteria.ACCURACY_HIGH, Criteria.ACCURACY_MEDIUM либо Criteria.ACCURACY_LOW.

- setBearingRequired. Если этот аргумент имеет значение true, это означает, что требуются данные о направлении движения пользователя. Эти данные применяются для ориентации карты, в результате чего направление движения пользователя указывает вверх экрана устройства.
- setCostAllowed. Если этот аргумент имеет значение true, это означает, что приложение может использовать службы передачи данных (например, подключение к Интернету для устройства), доступ к которым предоставляется на платной основе. Если вы планируете распространять подобное приложение, придется получить согласие пользователя на предоставление платных услуг.
- setPowerRequirement. Поставщики данных о местоположении потребляют различную мощность аккумуляторной батареи во время передачи данных о локации приложению. Аргумент Criteria.POWER_LOW определяет возврат приложения к провайдеру локации, который использует минимум мощности батареи во время предоставления требуемых для приложения данных. Также доступны аргументы Criteria.NO_REQUIREMENT, Criteria.POWER_HIGH и Criteria.POWER_MEDIUM.
- setAltitudeRequired. Если значение этого аргумента равно false, это означает, что приложение не требует данные о высоте.

Листинг 11.5. Переопределение методов onStart и onStop класса Activity

```
// вызывается, если Activity становится видимым пользователю
@Override
public void onStart()
{
   super.onStart(); // вызов метода onStart суперкласса

   // создание объекта Criteria, определяющего настройки
   // провайдера локации
   Criteria criteria = new Criteria();
   criteria.setAccuracy(Criteria.ACCURACY_FINE); // точные
                                          // данные локации
   criteria.setBearingRequired(true); // нужны данные сенсоров
                              // ускорения для вращения карты
   criteria.setCostAllowed(true); // согласие с платными услугами
   criteria.setPowerRequirement(Criteria.POWER_LOW); // попытка
                                                // экономии
   criteria.setAltitudeRequired(false); // не нужны данные о высоте

   // доступ к LocationManager
    locationManager =
      (LocationManager) getSystemService(LOCATION_SERVICE);

   // регистрация слушателя, определяющего «захват» GPS
```

```
102       locationManager.addGpsStatusListener(gpsStatusListener);
103
104       // доступ к лучшему провайдеру на основе значения Criteria
105       String provider = locationManager.getBestProvider(criteria, true);
106
107       // прослушивание изменений локации как можно чаще
108       locationManager.requestLocationUpdates(provider, 0, 0,
109          locationListener);
110
111       // доступ к диспетчеру питания устройства
112       PowerManager powerManager =
113          (PowerManager) getSystemService(Context.POWER_SERVICE);
114
115       // блокировка перехода в режим сна
116       wakeLock = powerManager.newWakeLock(
117          PowerManager.PARTIAL_WAKE_LOCK, "No sleep");
118       wakeLock.acquire(); // блокировка перехода в режим сна
119
120       bearingFrameLayout.invalidate(); // повторное отображение
                                           // BearingFrameLayout
121    } // завершение описания метода onStart
122
123    // вызывается, если Activity не отображается для пользователя
124    @Override
125    public void onStop()
126    {
127       super.onStop(); // вызов метода суперкласса
128       wakeLock.release(); // освобождение блокировки перехода в режим сна
129    } // конец описания метода onStop
130
```

В строках 98–99 выполняется доступ к системной службе LocationManager, которая назначается переменной экземпляра класса locationManager. В строке 102 регистрируется слушатель gpsStatusListener (см. листинг 11.8) в качестве GpsStatus.Lisener для LocationManager. С помощью этого слушателя определяется наличие «захвата» GPS. В этом состоянии устройство получает сигналы со спутников GPS, количество которых достаточно для определения местоположения.

Метод getBestProvider класса LocationManager (строка 105) возвращает значение типа String, представляющее имя поставщика данных о местоположении, который наилучшим образом соответствует критериям, заданным с помощью аргументов Criteria. Значение аргумента true свидетельствует о том, что следует вернуться к одному подходящему провайдеру.

Метод requestLocationUpdates класса LocationManager вызывается для регистрации слушателя locationListener (см. листинг 11.7), применяемого для прослушивания изменений локации, принимаемых от указанного провайдера. Если в качестве значения второго аргумента передается 0 (минимальное значение времени, выраженное в миллисекундах между изменениями локации), а также используется третий аргумент (минимальное расстояние в метрах, пройденное в период времени, соответствующий изменениям локации), это означает, что обновление локации осуществляется

с максимально возможной частотой (используется только в демонстрационных целях). Для каждого из значений аргументов могут использоваться положительные значения, которые способствуют экономии энергии батареи. Для выполнения «захвата» GPS обычно требуется несколько минут. Поэтому многие GPS-приложения используют метод getLastKnownLocation класса LocationManager для получения сведений о локации, которая была определена во время предыдущего «захвата» GPS (например, во время предыдущего выполнения приложения). Большинство людей практически все время находятся в пределах одного и того же географического региона, поэтому отображаемая на экране устройства карта будет довольно точно отображать реальное местонахождение пользователя.

В строках 112–113 осуществляется доступ к системной службе PowerManager. Метод newWakeLock класса PowerManager возвращает новый объект WakeLock (строки 116–117). С помощью метода acquire класса WakeLock (строка 118) гарантируется, что устройство остается на уровне потребления энергии, определенном WakeLock (как минимальный) до тех пор, пока не будет вызван метод release, который восстанавливает обычный уровень потребления энергии. Это приложение использует константу PowerManager.PARTIAL_WAKE_LOCK, определяющую, что приложение будет продолжать использовать вычислительную мощность CPU, даже если пользователь нажал кнопку питания устройства. Эта константа также может инициировать гашение экрана и отключение устройства. В результате приложение будет продолжать выполняться до тех пор, пока пользователь не коснется компонента ToggleButton Stop Tracking. Информация о различных доступных WakeLocks и их влиянии на потребление энергии батареи доступна на веб-сайте developer.android.com/reference/android/os/PowerManager.html.

Метод onStop (строки 124–130) вызывает метод release класса WakeLock, освобождающий блокировку пробуждения. Этот метод «снимает» блокировку перехода в режим сна и восстанавливает обычный режим потребления энергии батареи.

Метод updateLocation

Метод updateLocation (см. листинг 11.6), вызываемый с помощью LocationListener (см. листинг 11.7), получает объект Location и обновляет карту и слой соответствующим образом. Если данная локация не пуста и имеет место состояние «захвата» GPS, можно выполнить одно из следующих действий:

- Вызвать метод addPoint класса routeOverlay, который добавляет в маршрут выбранную локацию.
- Если имеется объект previousLocation, используется метод distanceTo класса Location (строка 143) для вычисления расстояния между текущей локацией и объектом previousLocation и вычисленное значение добавляется к значению distanceTraveled, которое отображается после завершения отслеживания маршрута пользователем.

Листинг 11.6. Метод updateLocation класса RouteTracker

```
131   // обновление локации на карте
132   public void updateLocation(Location location)
133   {
134      if (location != null && gpsFix) // локация не пустая; «захват» GPS
135         {
```

```
136           // добавление текущей локации к маршруту
137           routeOverlay.addPoint(location);
138
139           // при наличии предыдущей локации
140           if (previousLocation != null)
141           {
142              // добавление к сумме distanceTraveled
143              distanceTraveled += location.distanceTo(previousLocation);
144           }  // конец блока if
145
146           // получение значений широты и долготы
147           Double latitude = location.getLatitude() * 1E6;
148           Double longitude = location.getLongitude() * 1E6;
149
150           // создание точки GeoPoint, представляющей данную локацию
151           GeoPoint point =
152              new GeoPoint(latitude.intValue(), longitude.intValue());
153
154           // перемещение карты в текущую локацию
155           mapController.animateTo(point);
156
157           // обновление показаний компаса
158           bearingFrameLayout.setBearing(location.getBearing());
159           bearingFrameLayout.invalidate(); // перерисование
                                // на основании изменения положения
160        }  // конец блока if
161
162        previousLocation = location;
163     }    // конец описания метода updateLocation
164
```

- Получите значение широты и долготы локации и преобразуйте его в GeoPoint (строки 147–152). Значение GeoPoint состоит из значений *долготы* и *широты*, выраженных в микроградусах (миллионные доли градуса). С помощью методов getLatitude и getLongitude класса Location значения широты и долготы считываются в градусах. Чтобы преобразовать эти значения в градусы, каждое из них следует умножить на 1E6. Полученные результаты присваиваются переменным latitude и longitude соответственно. Потом эти новые значения применяются для создания объекта GeoPoint с целочисленными координатами.
- Метод animateTo класса MapController (строка 155) перемещает центр карты в выбранную точку GeoPoint с помощью *сглаженной анимации*. Если требуется извещение о завершении анимации, передайте этому методу объект Message либо Runnable.
- Метод getBearing класса Location (строка 158) применяется для получения сведений о направлении движения из последней локации. Направление движения передается в виде числа, значение которого равно количеству градусов в направлении на восток от истинного севера. Затем используется метод setBearing класса bearingFrameLayout, который обновляет направление движения, в результате чего карта может вращаться соответствующим образом. При этом используется метод invalidate класса bearingFrameLayout для перерисовывания карты.

ПРИМЕЧАНИЕ

> Направление движения можно также получить путем вызова метода bearingTo с аргументами Location, относящимися к предыдущей и текущей локации. Благодаря этому обеспечивается возможность вращения карты даже при тестировании приложения с помощью виртуального устройства AVD.

Независимо от того, была ли пустой локация, можно сохранить ее в виде предыдущей локации, чтобы подготовиться к приему значения следующей локации.

Анонимный класс LocationListener, отвечающий на события LocationManager

В листинге 11.7 определяется класс LocationListener. Объекты LocationListener получают события класса LocationManager, если изменяется физическое местоположение устройства либо изменяется статус провайдера локации. Это свойство активизируется путем вызова requestLocationUpdates (см. листинг 11.5, строки 108–109). Метод onLocationChanged (строки 170–176) вызывается в том случае, если устройство получает обновленный объект Location. Переменной gpsFix присваивается значение true в случае, если при получении объектов Location устройство «привязывается» к достаточно большому количеству спутников GPS для определения текущего местоположения пользователя. Если приложение в настоящее время отслеживает маршрут, вызывается метод updateLocation (см. листинг 11.6), с помощью которого к маршруту добавляется новый объект Location. Для этого приложения мы поддерживаем «пустые» методы, которые «отвечают» на изменения в статусе провайдеров локации (onProviderDisabled, onProviderEnabled и onStatusChanged). Если приложение должно «отвечать» на эти события, следует определить соответствующие методы.

Листинг 11.7. Слушатель LocationListener, отвечающий на события LocationManager

```
165 // обработка событий из LocationManager
166 private final LocationListener locationListener =
167    new LocationListener()
168 {
169    // если изменяется локация
170    public void onLocationChanged(Location location)
171    {
172       gpsFix = true; // после получения локаций имеем «захват» GPS
173
174       if (tracking) // отслеживание активно
175       updateLocation(location); // обновление локации
176    }  // конец описания onLocationChanged
177
178    public void onProviderDisabled(String provider)
179    {
180    }  // конец описания onProviderDisabled
181
182    public void onProviderEnabled(String provider)
183    {
184    }  // конец описания onProviderEnabled
```

```
185
186       public void onStatusChanged(String provider,
187          int status, Bundle extras)
188       {
189       }  // конец описания onStatusChanged
190    }; // конец описания locationListener
191
```

Анонимный внутренний класс, реализующий слушатель GpsStatus.Listener, который обрабатывает события GpsStatus

В листинге 11.8 определяется анонимный внутренний класс, реализующий интерфейс GpsStatus.Listener. Этот интерфейс позволяет определить момент приема устройством первого набора данных GPS. Отслеживание маршрута следует начинать только после перехода в режим приема данных GPS («захвата»), иначе вы не сможете добиться необходимой точности. В строке 197 с помощью GpsStatus.GPS_EVENT_FIRST_FIX определяется, имело ли место событие. Если событие имело место, переменной gpsFix присваивается значение true, и отображается сообщение Toast о том, что устройство захватило достаточное количество спутников GPS, необходимых для определения локации пользователя. Если на устройстве запущено другое приложение, которое запустило службу GPS и получило первые данные, наше приложение не получит событие первого «захвата». Именно поэтому в строке 172 переменной gpsFix присваивается значение true.

Листинг 11.8. Анонимный внутренний класс, реализующий слушатель GpsStatus.Listener для обеспечения возможности получать событие «захвата» GPS, которая позволит начать получать точные GPS-данные

```
192    // определение «захвата» GPS
193    GpsStatus.Listener gpsStatusListener = new GpsStatus.Listener()
194    {
195       public void onGpsStatusChanged(int event)
196       {
197          if (event == GpsStatus.GPS_EVENT_FIRST_FIX)
198          {
199             gpsFix = true;
200             Toast results = Toast.makeText(RouteTracker.this,
201                getResources().getString(R.string.toast_signal_acquired),
202                Toast.LENGTH_SHORT);
203
204             // центрирование объекта Toast на экране
205             results.setGravity(Gravity.CENTER,
206                results.getXOffset() / 2, results.getYOffset() / 2);
207             results.show(); // отображение результатов
208          }  // конец блока if
209       }  // конец определения метода GpsStatusChanged
210    }; // конец определения анонимного внутреннего класса
211
```

Переопределение метода isRouteDisplayed класса MapActivity

В листинге 11.9 переопределяется метод isRouteDisplayed класса MapActivity, который теперь будет возвращать значение false. Если разработанное вами приложение отображает информацию о маршруте, например направление движения, в соответствии с условиями пользовательского соглашения Google (Terms of Use) этот метод возвращает значение true. Предложение о соглашении с условиями этого соглашения отобразится при регистрации ключа API (code.google.com/android/add-ons/google-apis/mapkey.html).

Листинг 11.9. Переопределение метода isRouteDisplayed класса MapActivity

```
212   // В соответствии с соглашениями об использовании Google этот метод
213   // возвращает true, если отображаются сведения о маршруте, например
      // подсказки
214   @Override
215   protected boolean isRouteDisplayed()
216   {
217      return false; // информация о маршруте не отображается
218   }  // конец определения метода isRouteDisplayed
219
```

Переопределение методов onCreateOptionsMenu и onOptionsItemSelected класса Activity

В листинге 11.10 переопределяются методы onCreateOptionsMenu и onOptionsItemSelected класса Activity. Метод onCreateOptionsMenu использует объект MenuInflater для создания меню приложения на основе файла ресурсов меню route_tracker_menu.xml. Как только пользователь выберет один из пунктов меню, метод onOptionsItemSelected обработает соответствующее событие. Если пользователь выбирает пункт меню MenuItem Map, в строке 238 вызывается метод setSatellite класса MapView с аргументом false, который определяет отображение стандартной карты. Если же пользователь выбирает пункт меню MenuItem Satellite, в строке 241 вызывается метод setSatellite с аргументом true, определяющий отображение спутниковой фотографии.

Листинг 11.10. Переопределение методов onCreateOptionsMenu и onOptionsItemSelected класса Activity

```
220   // создание меню Activity на основе XML-файла ресурсов меню
221   @Override
222   public boolean onCreateOptionsMenu(Menu menu)
223   {
224      super.onCreateOptionsMenu(menu);
225      MenuInflater inflater = getMenuInflater();
226      inflater.inflate(R.menu.route_tracker_menu, menu);
227      return true;
228   }  // конец описания метода onCreateOptionsMenu
229
230   // обработка параметров, выбранных в меню
231   @Override
232   public boolean onOptionsItemSelected(MenuItem item)
233   {
```

```
      // выполнение соответствующей задачи
      switch (item.getItemId())
      {
         case R.id.mapItem: // если пользователь выбрал "Map"
            mapView.setSatellite(false); // отображение карты
            return true;
         case R.id.satelliteItem: // если пользователь выбрал "Satellite"
            mapView.setSatellite(true); // спутниковая фотография
            return true;
         default:
            return super.onOptionsItemSelected(item);
      } // конец блока switch
   } // конец описания метода onOptionsItemSelected

```

Анонимный внутренний класс, который реализует слушатель OnCheckedChangeListener, отслеживающий события trackingToggleButton

В листинге 11.11 определяется слушатель trackingToggleButtonListener класса OnCheckedChangeListener, обрабатывающий события метода trackingToggleButton путем отображения результатов для завершенного маршрута либо начала отслеживания нового маршрута.

Листинг 11.11. Слушатель trackingToggleButtonListener, отслеживающий события trackingToggleButton

```
   // слушатель событий из trackingToggleButton
   OnCheckedChangeListener trackingToggleButtonListener =
      new OnCheckedChangeListener()
   {
      // вызывается, если пользователь выбрал состояние отслеживания
      @Override
      public void onCheckedChanged(CompoundButton buttonView,
         boolean isChecked)
      {
         // если приложение отслеживается
         if (!isChecked)
         {
            tracking = false; // прекращено отслеживание локаций

            // вычисление общего времени отслеживания
            long milliseconds = System.currentTimeMillis() – startTime;
            double totalHours = milliseconds / MILLISECONDS_PER_HOUR;

            // создание диалогового окна отображения результатов
            AlertDialog.Builder dialogBuilder =
               new AlertDialog.Builder(RouteTracker.this);
            dialogBuilder.setTitle(R.string.results);

            double distanceKM = distanceTraveled / 1000.0;
```

продолжение ☞

Листинг 11.11 *(продолжение)*

```
272            double speedKM = distanceKM / totalHours;
273            double distanceMI = distanceKM * MILES_PER_KILOMETER;
274            double speedMI = distanceMI / totalHours;
275
276            // отображение пройденного расстояния distanceTraveled
               // и средней скорости
277            dialogBuilder.setMessage(String.format(
278               getResources().getString(R.string.results_format),
279               distanceKM, distanceMI, speedKM, speedMI));
280            dialogBuilder.setPositiveButton(
281               R.string.button_ok, null);
282            dialogBuilder.show(); // display the dialog
283         }  // конец блока if
284         else
285         {
286            tracking = true; // отслеживается приложение
287            startTime = System.currentTimeMillis(); // текущее время
288            routeOverlay.reset(); // переустановка нового маршрута
289            bearingFrameLayout.invalidate(); // очистка маршрута
290            previousLocation = null; // запуск нового маршрута
291         }  // конец блока else
292      }  // конец определения метода onCheckChanged
293   }; // конец определения анонимного внутреннего класса
294 } // конец определения класса RouteTracker
```

Если пользователь выбирает кнопку trackingToggleButton, вызывается метод onCheckedChanged, в качестве второго аргумента которого используется текущее состояние кнопки. Если кнопка не выбрана (строка 258), приложение не выполняет отслеживание, поэтому в строках 260–282 вычисляются и отображаются результаты отслеживания. В строках 263–264 определяются переменные totalHours, включающие значение времени, в течение которого выполнялось отслеживание маршрута. Эти значения можно использовать для определения скорости перемещения пользователя. Переменная distanceTraveled представляет пройденное расстояние (в метрах). Это расстояние делится на 1000.0 (строка 271) для определения пройденного расстояния в километрах. В строке 272 определяется скорость (км/час). В строках 273–274 вычисляется пройденное расстояние в милях, а также скорость, выраженная в милях/час.

Выбор кнопки trackingToggleButton при возникновении события означает, что пользователь начал отслеживать маршрут. В этом случае в строках 286–290 отображается сообщение о начале отслеживания, получается начальное время маршрута, переопределяется routeOverlay, вызывается метод invalidate класса bearingFrameLayout (для очистки предыдущего маршрута при его наличии), а переменной previousLocation присваивается значение null. При выборе параметра Stop Tracking переменной tracking присваивается значение false (строка 282), чтобы указать отсутствие отслеживания. Мы вычисляем время отслеживания totalMilliseconds путем вычитания значения переменной startTime из значения, возвращаемого объектом System.currentMillis.

11.5.2. Подкласс BearingFrameLayout класса FrameLayout

Класс BearingFrameLayout (см. листинги 11.12–11.16) поддерживает компонент MapView приложения и ориентирует его таким образом, чтобы текущее направление движения пользователя совпадало с направлением на верхний край экрана устройства.

Операторы package, import и переменные экземпляра класса

В листинге 11.12 приведены операторы package, import и переменные экземпляра BearingFrameLayout. Переменная экземпляра класса scale может использоваться для увеличения высоты и ширины компонента MapView в соответствии с размерами экрана. В результате обеспечивается заполнение картой экрана при повороте устройства.

Листинг 11.12. Операторы package, import и переменные экземпляра класса BearingFrameLayout

```
// BearingFrameLayout.java
// Вращение MapView в соответствии с направлением движения устройства.
package com.deitel.routetracker;

import com.google.android.maps.MapView;

import android.app.Activity;
import android.content.Context;
import android.graphics.Canvas;
import android.view.Display;
import android.widget.FrameLayout;

public class BearingFrameLayout extends FrameLayout
{
    private int scale = 0;     // величина масштабирования разметки
    private MapView mapView;   // отображение карт Google
    private float bearing = 0f; // направление стрелки компаса

```

Метод getChildLayoutParams

В листинге 11.13 определяется метод getChildLayoutParams, возвращающий объект LayoutParams. Этот объект представляет порядок размещения дочернего компонента View в родительском макете. Объекты LayoutParams являются специфичными для компонентов Views и ViewGroups. Например, компонент LinearLayouts использует подкласс класса LayoutParams, который отличается от подкласса, применяемого компонентом RelativeLayouts. В пользовательских компонентах View могут определяться собственные классы LayoutParams, использующие пользовательские параметры. Для настройки различных параметров разметки, определяющих использование XML, указываются значения, такие как match_parent либо wrap_content для параметров width и/или height компонентов View графического интерфейса пользователя.

Листинг 11.13. Метод getChildLayoutParams класса BearingFrameLayout

```
19   // возвращает параметры разметки для MapView
20   public LayoutParams getChildLayoutParams()
21  {
22      Display display =
23         ((Activity) getContext()).getWindowManager().getDefaultDisplay();
24      int w = display.getWidth();
25      int h = display.getHeight();
26      scale = (int) Math.sqrt((w * w) + (h * h));
27
28      return new LayoutParams(scale, scale);
29   }  // конец описания метода getChildLayoutParams
30
```

В строках 22–23 открывается доступ к заданному по умолчанию системному объекту Display, представляющему экран устройства. Класс Display поддерживает *размер*, *частоту обновления* и *текущую ориентацию* экрана. Методы getWidth и getHeight возвращают размеры экрана. Для компонента BearingMapView следует выбрать размеры, которые являются достаточно большими, чтобы заполнить весь экран при вращении MapView в соответствии с выбранным направлением движения пользователя. Для этого нужно изменить масштаб MapView таким образом, чтобы значения ширины и высоты соответствовали диагонали экрана, которая вычисляется в строке 26. Если этого не сделать, то в процессе вращения MapView будут появляться черные области по углам экрана устройства (из-за прямоугольной формы элементов карты).

Конструктор

В листинге 11.14 определяется конструктор класса BearingFrameLayout. Мы вызываем конструктор суперкласса, передаем ему содержимое. После создания нового компонента MapView передаем ему apiKey Google Maps. В строках 37–43 конфигурируется компонент MapView следующим образом:

- setClickable. Если этому аргументу присвоено значение true, пользователь может взаимодействовать с соответствующим компонентом MapView, выполняя масштабирование и панорамирование. Предварительно нужно активизировать MapView.
- setEnabled. После присваивания значения true этому аргументу компонент MapView активизируется. Если компонент не активен, пользователь не может взаимодействовать с картой, касаясь ее.
- setSatellite. Изначально этому аргументу присвоено значение false, что приводит к отображению стандартных карт Google, а не спутниковых фотографий.
- setBuiltInZoomControls. Если этому аргументу присвоено значение true, активизируются встроенные элементы управления масштабом MapView.
- setLayoutParams. Аргумент LayoutParams определяет, каким образом будет конфигурироваться MapView в его родительской разметке. В данном случае этот аргумент используется для определения размеров MapView.

В строке 44 добавляется объект mapView в качестве потомка объекта BearingFrameLayout.

Листинг 11.14. Конструктор класса BearingFrameLayout

```
// общедоступный конструктор класса BearingFrameLayout
public BearingFrameLayout(Context context, String apiKey)
{
   super(context); // вызов конструктора суперкласса

   mapView = new MapView(context, apiKey); // создание нового MapView
   mapView.setClickable(true); // обеспечение взаимодействия
                               // пользователя с картой
   mapView.setEnabled(true); // активизация генерирования событий
                             //компонентом MapView
   mapView.setSatellite(false); // отображение традиционной карты
   mapView.setBuiltInZoomControls(true); // активизация элементов
                                         // управления масштабом

   // настройка макета MapView
   mapView.setLayoutParams(getChildLayoutParams());
   addView(mapView); // добавление MapView в макет
} // конец определения конструктора BearingFrameLayout

```

Переопределение метода dispatchDraw класса View

В листинге 11.15 переопределяется метод dispatchDraw класса View. Этот метод вызывается с помощью родительского метода draw класса View для отображения дочерних компонентов View. Можете переопределить этот метод для контроля отображения дочерних компонентов View. Путем поворота компонента View можно добиться соответствия с показаниями компаса.

Листинг 11.15. Переопределение метода dispatchDraw класса View

```
// вращение карты в соответствии с направлением движения пользователя
@Override
protected void dispatchDraw(Canvas canvas)
{
   if (bearing >= 0) // если значение bearing больше 0
   {
     // получение размеров холста
     int canvasWidth = canvas.getWidth();
     int canvasHeight = canvas.getHeight();

     // размеры масштабированного холста
     int width = scale;
     int height = scale;

     // центрирование масштабированного холста
     int centerXScaled = width / 2;
     int centerYScaled = height / 2;

     // центр холста на экране
     int centerX = canvasWidth / 2;
```

продолжение ☞

Листинг 11.15 *(продолжение)*

```
67        int centerY = canvasHeight / 2;
68
69        // перемещение центра масштабированной области в фактический
          // центр экрана
70        canvas.translate(-(centerXScaled — centerX),
71           -(centerYScaled — centerY));
72
73        // вращение вокруг центра экрана
74        canvas.rotate(-bearing, centerXScaled, centerYScaled);
75      }  // конец блока if
76
77        super.dispatchDraw(canvas); // рисование дочерних
                                      // представлений для данного макета
78    }  // конец метода dispatchDraw
79
```

В строках 54–55 определяются размеры поверхности, доступной для рисования (размеры выбранного объекта Canvas). Потом выполняется масштабирование размеров путем умножения на определенное число с помощью метода getLayoutParams и вычисление центральных точек исходных и масштабированных размеров (строки 58–67).

ПРИМЕЧАНИЕ

Масштабирование карт *не разрешается* в терминах обслуживания Google — это операция выполняется лишь в демонстрационных целях. Существует ряд других API карт, для которых имеют место свои условия обслуживания.

Затем выполняется перемещение центральной точки холста на величину, равную расстоянию между двумя точками. (При этом используются масштабированные размеры для параметров макета компонента View, строки 70–71.) Затем выполняется поворот объекта Canvas вокруг новой центральной точки на –bearing градусов (строка74). Как вы помните, параметр bearing представляет направление движения пользователя (в градусах) в восточном направлении от истинного севера. Если направление на истинный север указывает на верхнюю часть устройства, а вы начинаете перемещаться на северо-восток, направление движения будет представлено положительным числом градусов, которое определяет перемещение в сторону правого верхнего угла устройства. В этом случае карта должна повернуться влево на угол, равный количеству градусов. Именно поэтому знак угла поворота меняется на противоположный. Вращение объекта Canvas с помощью метода dispatchDraw приводит к рисованию в данном компоненте View, включая объект Overlay, представляющий маршрут. При этом вращение осуществляется таким образом, что карта была сориентирована в направлении движения пользователя. В строке 77 выполняется рисование всех других дочерних компонентов View.

Методы setBearing и getMapView

В листинге 11.16 определяются методы setBearing и getMapView класса BearingFrameLayout. Метод setBearing определяет вращение объекта в соответствии со значением аргумента,

а метод getMapView возвращает компонент MapView. Эти методы относятся к классу RouteTracker.

Листинг 11.16. Методы setBearing и MapView класса BearingFrameLayout

```
   // настройка компаса
   public void setBearing(float bearing)
   {
      this.bearing = bearing;
   }  // конец определения метода setBearing

   // возврат MapView
   public MapView getMapView()
   {
      return mapView;
   }  // конец определения метода getMapView
} // конец определения класса BearingFrameLayout
```

11.5.3. Подкласс RouteOverlay класса Overlay

Подкласс Overlay класса RouteOverlay (см. листинги 11.17–11.20) поддерживает отслеженные данные Location и рисует маршрут.

Операторы package, import и переменные экземпляра класса

В листинге 11.17 приведены операторы package, import и переменные экземпляра класса RouteOverlay. Константа POSITION_MARKER определяет частоту отображения черных точек вдоль маршрута пользователя.

Листинг 11.17. Операторы package, import и переменные экземпляра класса

```
// RouteOverlay.java
// Рисование маршрута на MapView.
package com.deitel.routetracker;

import java.util.ArrayList;
import java.util.List;

import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.Point;
import android.location.Location;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;

public class RouteOverlay extends Overlay
{
```

продолжение ⇨

Листинг 11.17 *(продолжение)*

```
    private List<Location> locations; // данные отслеживания Location
    private Paint pathPaint; // информация Paint для Path
    private Paint positionPaint; // информация Paint для текущей позиции
    private final int POSITION_MARKER = 10; // маркер частоты

```

Конструктор класса RouteOverlay

В листинге 11.18 определяется конструктор класса RouteOverlay. В строках 29–33 определяется объект Paint, задающий настройки рисуемой линии, которая представляет маршрут. В результате вызова метода setAntiAlias класса Paint активизируется эффект *сглаживания*, применяемый для округления краев линии. Выбирается красный цвет, стиль STROKE и толщина линии, равная 5. Массив ArrayList<Location>, вызываемый локациями (строка 34), хранит объекты Locations наравне с отслеженным маршрутом. В строках 37–39 конфигурируется второй объект Paint, используемый для отображения черных кружков для каждого POSITION_MARKER, определяющего число локаций.

Листинг 11.18. Конструктор класса RouteOverlay

```
26   public RouteOverlay()
27  {
28      // объект Paint, предназначенный для рисования объекта Path
        // красной линией толщиной 5
29      pathPaint = new Paint();
30      pathPaint.setAntiAlias(true);
31      pathPaint.setColor(Color.RED);
32      pathPaint.setStyle(Paint.Style.STROKE);
33      pathPaint.setStrokeWidth(5);
34      locations = new ArrayList<Location>(); // инициализация точек
35
36      // объект Paint, применяемый для рисования черных кружков
        // для каждой из локаций, заданных POSITION_MARKER
37      positionPaint = new Paint();
38      positionPaint.setAntiAlias(true);
39      positionPaint.setStyle(Paint.Style.FILL);
40   }  // конец определения конструктора RouteOverlay
41
```

Методы addPoint и reset

В листинге 11.19 определяются методы addPoint и reset. Каждый раз, когда RouteTracker принимает событие о новой локации, он передает Location методу addPoint. Этот метод добавляет локацию в массив ArrayList<Location>. Метод reset вызывается RouteTracker и применяется для очистики списка предыдущих локаций перед началом отслеживания нового маршрута.

Листинг 11.19. Методы addPoint и reset класса RouteOverlay

```
// добавление новой локации в список локаций
public void addPoint(Location location)
```

```
{
   locations.add(location);
}  // конец определения метода addPoint

// переустановка Overlay для отслеживания нового маршрута
public void reset()
{
   locations.clear(); // удаление всех предыдущих локаций
}  // конец описания метода reset

```

Переопределение метода Overlay

В листинге 11.20 переопределяется метод draw класса Overlay, используемый для рисования отслеженного маршрута в области компонента MapView. Этот метод принимает объект Canvas (холст), MapView (mapView) и булеву переменную shadow. В этот момент вызывается метод draw суперкласса. При первом вызове этого метода в качестве последнего аргумента передается значение true. Класс Overlay рисует слой shadow, затем метод вызывается снова с аргументом false для рисования слоя overlay. На слое shadow обычно отображаются тени элементов, например маркеры карты, которые отображает Google в процессе поиска с помощью Google Maps.

Листинг 11.20. Переопределение метода draw класса View

```
// рисование слоя Overlay в верхней части MapView
@Override
public void draw(Canvas canvas, MapView mapView, boolean shadow)
{
   super.draw(canvas, mapView, shadow); // вызов метода
                                        // draw суперкласса
   Path newPath = new Path(); // получение нового объекта Path
   Location previous = null;  // инициализация предыдущей локации нулем

   // для каждой локации
   for (int i = 0; i < locations.size(); ++i)
   {
      Location location = locations.get(i);

      // преобразование Location в GeoPoint
      Double newLatitude = location.getLatitude() * 1E6;
      Double newLongitude = location.getLongitude() * 1E6;
      GeoPoint newPoint = new GeoPoint(newLatitude.intValue(),
         newLongitude.intValue());

      // преобразование GeoPoint в точку на экране
      Point newScreenPoints = new Point();
      mapView.getProjection().toPixels(newPoint, newScreenPoints);

      if (previous != null) // если это не первая локация
      {
         // получение GeoPoint для предыдущей локации
         Double oldLatitude = previous.getLatitude() * 1E6;
```

продолжение ⇨

Листинг 11.20 *(продолжение)*

```
81             Double oldLongitude = previous.getLongitude() * 1E6;
82             GeoPoint oldPoint = new GeoPoint(oldLatitude.intValue(),
83                oldLongitude.intValue());
84
85             // преобразование GeoPoint в точку на экране
86             Point oldScreenPoints = new Point();
87             mapView.getProjection().toPixels(oldPoint, oldScreenPoints);
88
89             // добавление новой точки в объект Path
90             newPath.quadTo(oldScreenPoints.x, oldScreenPoints.y,
91               (newScreenPoints.x + oldScreenPoints.x) / 2,
92               (newScreenPoints.y + oldScreenPoints.y) / 2);
93
94             // возможно рисование черной точки для текущей позиции
95             if ((i % POSITION_MARKER) == 0)
96                canvas.drawCircle(newScreenPoints.x, newScreenPoints.y, 10,
97                   positionPaint);
98          }  // конец блока if
99          else
100         {
101            // перемещение к первой локации
102            newPath.moveTo(newScreenPoints.x, newScreenPoints.y);
103         }  // конец блока else
104
105         previous = location; // хранение локации
106      }  // конец цикла for
107
108         canvas.drawPath(newPath, pathPaint); // рисование контура
109      }  // конец описания метода draw
110   }  // конец описания класса RouteOverlay
```

Мы рисовали маршрут в форме объекта Path, поэтому в строке 59 сначала создается новый объект Path. Затем предыдущему объекту Location присваивается значение null, поскольку объект Path перерисовывается при каждом вызове метода draw. Потом для каждого объекта Location в точках ArrayList<Location> выполняются следующие действия:

- Получаем новый обьект Location из имеющихся локаций (строка 65).
- Создаем точку GeoPoint для каждого объекта Location (строки 68–71) с помощью методики, описанной в листинге 11.6.
- Преобразуем точку GeoPoint для объекта Location в точку, отображенную на экране (строки 74–75). Метод getProjection класса MapView поддерживает объект Projection, выполняющий преобразования между пиксельными и географическими координатами. Этот метод следует использовать для получения обновленного объекта Projection, поскольку при каждом перерисовывании компонента MapView может изменяться объект Projection. Метод toPixels класса Projection использует аргументы GeoPoint и Point. Пиксельные координаты соответствуют месту на экране, в котором отображаемые значения широты и долготы GeoPoint вставляются в объект Point.

Если Location previous не равен null, мы подготавливаем новый сегмент линии для маршрута:

- В строках 80–87 выбирается новый объект GeoPoint для previous Location, который преобразуется в точку на экране.
- В строках 90–92 используется метод quadTo класса Path для добавления следующего сегмента линии в объект Path (в виде квадратичной кривой Безье).
- В строках 95–97 рисуется кривая, если текущий индекс Location (i) делится на константу POSITION_MARKER.

Если переменная previous равна null, мы обрабатываем первый объект Location в списке, поэтому в строке 102 просто используется метод moveTo класса Path для перемещения объекта Point, определенного с помощью newScreenPoints. В конце оператора for в строке 105 хранится текущая локация (в переменной previous), которая используется при следующей итерации цикла. После обработки всех объектов Location рисуется новый объект newPath на холсте.

11.6. Резюме

В этой главе было создано приложение Route Tracker, с помощью которого выполняется отслеживание перемещения пользователя, которое отображается в виде линии на карте Google Map. Приложение использует несколько новых функций, описанных в файле манифеста. Чтобы получить доступ к библиотеке Google Maps API, укажите название библиотеки в манифесте приложения с помощью элемента uses-library. Мы удалили строку заголовка Activity путем изменения темы Activity с помощью атрибута android:theme элемента activity, а также с помощью элементов uses-permission запрашивали разрешения на использование различных системных служб, требуемых для корректной работы приложения.

Переключатель ToggleButton обеспечивал поддержку состояния on–off, представляющего состояние отслеживания приложением маршрута пользователя. Обработка событий ToggleButton выполнялась с помощь реализации интерфейса CompoundButton.OnCheckedChangeListener.

Для организации взаимодействия с Google Maps API были использованы различные классы из пакета com.google.android.maps. Был расширен класс MapActivity, с помощью которого создана Activity, управляемая с помощью MapView. Для отображения данных посредством MapView использовался подкласс класса Overlay и был переопределен его метод draw. С помощью объектов GeoPoints преобразуются GPS-данные в точки, в которых повторно центрируется карта на основе текущей локации пользователя, а также рисуется маршрут пользователя.

При обработке данных локации использовались функции, определенные в пакете android.location. Класс LocationManager обеспечивает доступ к службам определения локации устройства и выбирает лучшего поставщика данных о местоположении, основываясь на требованиях, которые определены с помощью объекта Criteria. Затем у данного поставщика были запрошены обновления, которые были переданы LocationListener. Этот объект получает обновления в виде объектов Location, представляющих географическое

положение устройства. Для определения состояния «захвата» GPS для устройства был реализован интерфейс GpsStatus.Listener.

С помощью класса PowerManager приложение может управлять состоянием энергопотребления устройства, в результате чего фиксируются данные о локации устройства, даже если экран находится в отключенном состоянии. С помощью класса Display были получены сведения о размерах экрана устройства, затем выполнялось масштабирование карты. Это позволяет заполнять картой весь экран при повороте устройства.

В главе 12 будет создано приложение Slideshow, предназначенное для создания и отображения слайд-шоу на основе изображений и музыки. С помощью этого приложения пользователь сможет получить доступ к музыке и фотоальбомам, которые хранятся на устройстве Android. В слайд-шоу появится возможность добавить новые фотографии, а также выбрать песню, которая будет звучать во время демонстрации слайд-шоу.

Приложение Slideshow

12

Доступ к библиотекам Gallery и Media, встроенные поставщики Content Providers, плеер MediaPlayer, переходы между изображениями, пользовательские макеты Custom ListActivity и шаблон View-Holder

В этой главе...

- Использование объектов Intents и провайдеров контента для выбора изображений и музыки из галереи и медиатеки устройства соответственно.
- Запуск объектов Intents, возвращающих результат.
- Использование плеера MediaPlayer для воспроизведения музыки из медиатеки устройства во время просмотра слайд-шоу.
- Настройка макета ListActivity.
- Использование шаблона view holder для улучшения производительности при использовании сложных макетов, состоящих из элементов ListView.
- Создание пользовательского графического интерфейса для диалоговых окон AlertDialog, обеспечивающего ввод информации пользователем.
- Загрузка изображений в виде растров с помощью BitmapFactory.
- Использование метода TransitionDrawable для создания переходов между двумя объектами BitmapDrawable, содержащими изображения.

12.1. Введение

Приложение Slideshow позволяет создавать слайд-шоу и управлять ими. При этом используются изображения и музыка, находящиеся в галерее и музыкальной библиотеке устройства. На рис. 12.1 показано окно приложения, в которое было добавлено несколько слайд-шоу. В области ListView отображается заголовок и первый кадр слайд-шоу вместе с тремя кнопками Button. Для начала воспроизведения слайд-шоу коснитесь кнопки Play Button. Каждый кадр отображается в течение пяти секунд, причем в это время воспроизводится выбранная пользователем мелодия (в фоновом режиме). Между изображениями слайд-шоу выбран *переход перекрестное затухание*. После выбора кнопки Edit Button (Изменить) слайд-шоу отображается объект Activity, предназначенный для выбора изображений и музыки. Если коснуться кнопки Delete Button (Удалить), соответствующее слайд-шоу будет удалено. Эта версия приложения *не сохраняет* слайд-шоу после закрытия приложения — эта функция появится в приложении Enhanced Slideshow, которое будет создано в главе 13.

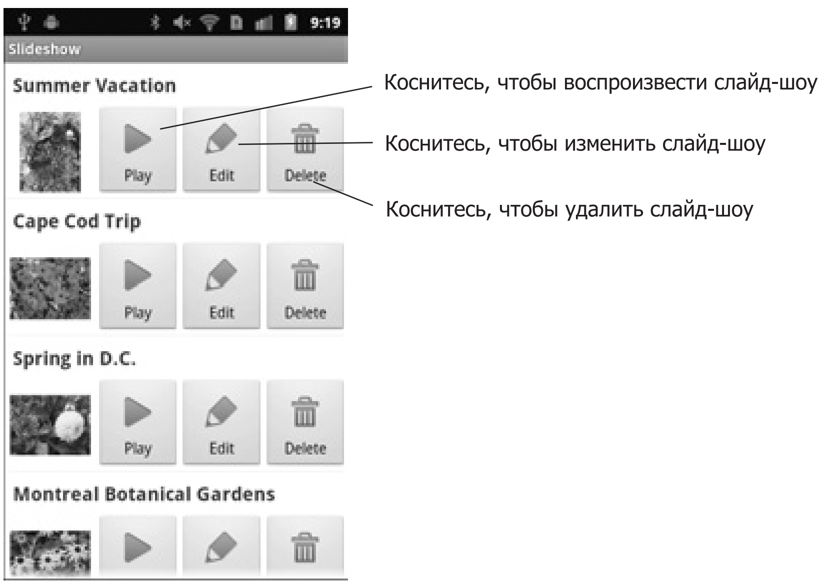
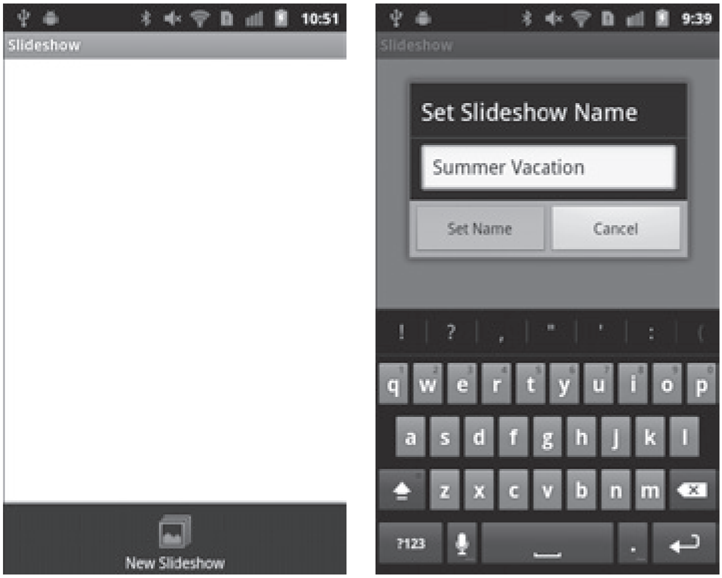


Рис. 12.1. Список слайд-шоу, созданных пользователем

Если приложение запускается впервые, список слайд-шоу будет пустым. После выбора кнопки меню устройства появится пункт меню New Slideshow (Новое слайд-шоу) (рис. 12.2, *а*). После выбора этого пункта меню отобразится диалоговое окно Set Slideshow Name (Выбрать название слайд-шоу) (рис. 12.2, *б*), в котором присваивается имя новому слайд-шоу. Если пользователь выберет кнопку Set Name (Выбрать имя) в диалоговом окне, создается новое слайд-шоу и отображается объект Slideshow Editor Activity (рис. 12.3).

После выбора пользователем пункта меню Add Picture (Добавить изображение) появится окно приложения Gallery устройства (рис. 12.4, *а*). Затем пользователь может



а *б*

Рис. 12.2. Добавление нового слайд-шоу и выбор его названия: *а* — после выбора кнопки меню устройства появится элемент меню New Slideshow; *б* — после выбора пункта меню New Slideshow в меню приложения появится диалоговое окно Set Slideshow Name (в этом окне пользователь ввел имя слайд-шоу и коснулся кнопки Set Name)

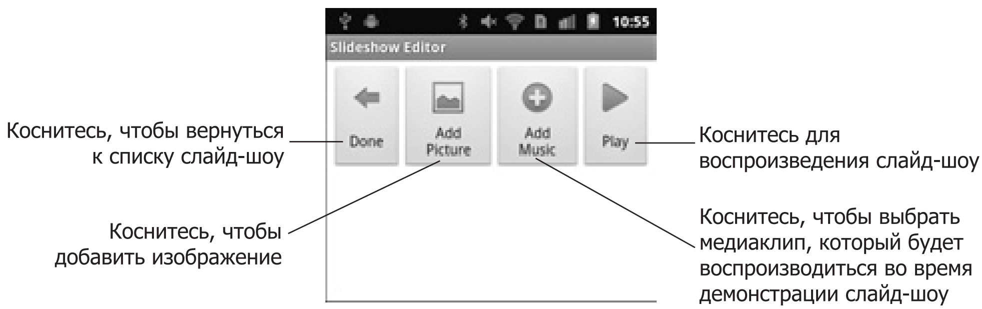
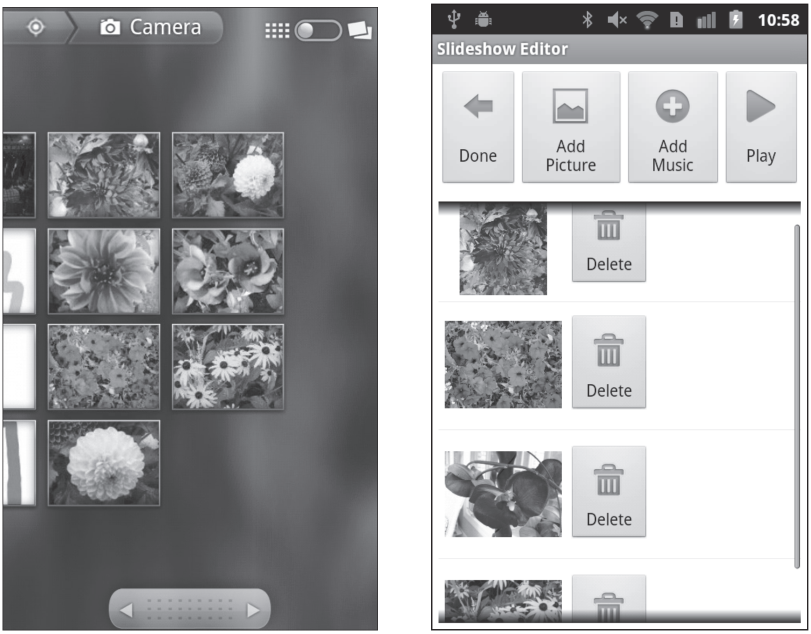


Рис. 12.3. Объект Slideshow Editor Activity до добавления каких-либо изображений в слайд-шоу

выбрать существующее изображение или сделать новую фотографию с помощью камеры устройства. Чтобы добавить фотографию в слайд-шоу, коснитесь ее. На рис. 12.4, *б* показано окно Slideshow Editor Activity после добавления нескольких изображений в слайд-шоу. Темные полосы, находящиеся в верхней и нижней части компонента ListView, свидетельствует о том, что могут отобразиться дополнительные изображения, для просмотра которых используется прокрутка. Кнопка Delete Button, находящаяся



а *б*

Рис. 12.4. Приложение Gallery, с помощью которого выбираются изображения, и окно Slideshow Editor Activity после выбора нескольких изображений рисунка: *а* — после выбора пользователем пункта меню Add Picture отобразится окно приложения Gallery устройства. Пользователь может выбрать изображение либо сделать новую фотографию с помощью камеры; *б* — окно Slideshow Editor Activity после добавления пользователем в слайд-шоу нескольких изображений

справа от каждого изображения, предназначена для удаления этого изображения из слайд-шоу.

После выбора пользователем пункта меню Add Music Button (Добавить музыку) Android отобразит список приложений, в котором пользователь может выбрать приложение Music (Музыка). При использовании типичного устройства Android отобразится диалоговое окно, в котором пользователю станет доступным диалоговое окно (рис. 12.5), где можно выбрать параметры Select music track (Выбор музыкальной дорожки) либо Sound Recorder (Запись звука). После выбора параметра Select music track отобразится список музыкальных записей, хранящихся на устройстве. После выбора пункта меню Sound Recorder будет запущено приложение Sound Recorder, в котором пользователь может создать новую запись, которая будет воспроизводиться во время демонстрации слайд-шоу. Если пользователь создал новую музыкальную запись, она также появится в списке музыкальных произведений устройства после отображения списка в следующий раз. Пользователь может просматривать слайд-шоу во время изменения. Для этого следует нажать кнопку Play в окне Slideshow Editor (либо в основном списке слайд-шоу). На рис. 12.6 показано одно изображение из воспроизводимого в настоящее время слайд-шоу.

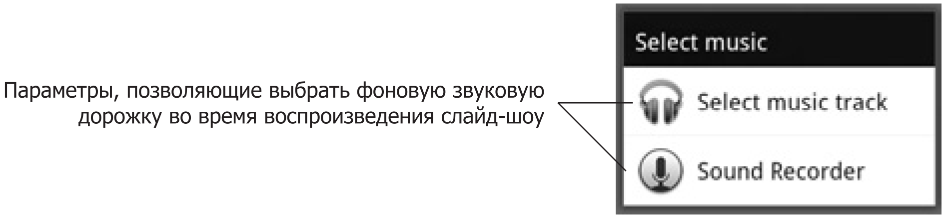


Рис. 12.5. Диалоговое окно выбора деятельности, отображаемое Android, в котором пользователь может выбрать существующий медиаклип (Select music track) или записать новый (Sound Recorder)

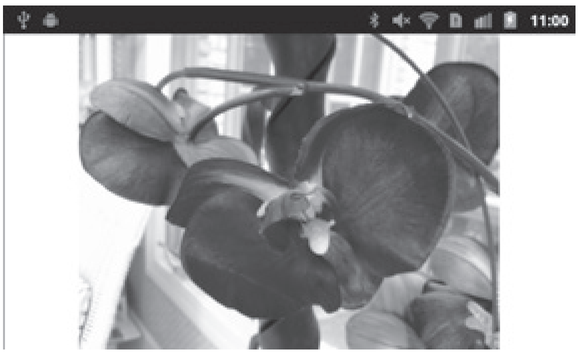


Рис. 12.6. Изображение, отображаемое во время воспроизведения слайд-шоу

12.2. Тестирование приложения Slideshow App

Открытие и выполнение приложения

Откройте среду Eclipse и импортируйте проект Slideshow app. Выполните следующие действия:

1. Выполните команды File ▸ Import… (Файл ▸ Импорт...) для отображения диалогового окна Import (Импорт).
2. Раскройте узел General (Общий) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область). Затем щелкните на кнопке Next > (Далее >).
3. Справа от текстового поля выберите параметр Select root directory (Выбрать корневой каталог) и щелкните на кнопке Browse… (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папки) выберите папку Slideshow в папке примеров книги и щелкните на кнопке OK.
4. Щелкните на кнопке Finish (Готово) для импорта проекта в среду Eclipse.

В среде Eclipse щелкните правой кнопкой мыши на проекте приложения, находящемся в окне Package Explorer, затем в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android).

Передача музыки и фотографий виртуальному устройству AVD

Для тестирования приложения Slideshow добавьте изображения и музыку на виртуальное устройство AVD. Поместите медиаресурсы на SD-карту устройства AVD, которая сконфигурирована при настройке AVD. Выполните следующие действия:

1. Запустите виртуальное устройство AVD с помощью Android SDK и AVD Manager.
2. В среде Eclipse откройте перспективу DDMS, выполнив команды Window ▸ Open Perspective (Окно ▸ Открыть перспективу).
3. На панели перспективы DDMS в списке Devices (Устройства) выберите ваше устройство AVD.
4. В правой части панели перспективы DDMS выберите вкладку File Explorer (Обозреватель файлов) для отображения файловой системы AVD.
5. Перейдите в папку /mnt/sdcard, потом в эту папку перетащите музыку и фотографии.
6. Закройте устройство AVD и перезапустите его, не выбирая параметр Launch from snapshot checked. В результате AVD просканирует карту памяти SD в поисках новых изображений и/или музыки.

Несколько примеров изображений цветов находятся в папке images в папке с примерами кодов книги. На многих веб-сайтах можно найти свободно загружаемые музыкальные файлы, подходящие для тестирования (файлы в формате MP3).

Добавление нового слайд-шоу

Коснитесь кнопки меню устройства потом коснитесь кнопки New Slideshow Button, чтобы открыть диалоговое окно Set Slideshow Name. Выберите имя слайд-шоу, затем выберите параметр Set Name для создания нового слайд-шоу и его отображения в окне Slideshow Editor.

Изменение нового слайд-шоу

Коснитесь кнопки Add Picture Button для просмотра содержимого приложения Gallery. Коснитесь фотографии в окне приложения Gallery, чтобы добавить ее в слайд-шоу. Повторяйте этот процесс для каждого изображения, которое вы хотите добавить в слайд-шоу. Если коснуться кнопки Back (Назад) устройства до прикосновения к фотографии, то вы вернетесь в окно Slideshow Editor без добавления фотографии. Если нужно удалить только что добавленное изображение, коснитесь кнопки Delete Button, которая находится справа от фотографии.

Для выбора фоновой музыки коснитесь кнопки Add Music Button (Добавить музыку). На экране появится окно с параметрами Select music track и Sound Recorder. Выберите параметр Select music track для выбора существующего музыкального файла или параметр Sound Recorder для записи вашего собственного музыкального файла. После выбора музыкальной звукозаписи вы вернетесь в окно Slideshow Editor.

Воспроизведение слайд-шоу

Для воспроизведения слайд-шоу воспользуйтесь одним из следующих методов:

- В окне Slideshow Editor коснитесь кнопки Play Button.
- Или коснитесь кнопки Done Button (Готово) в окне Slideshow Editor, чтобы вернуться к списку слайд-шоу, затем нажмите кнопку Play Button (Воспроизведение), находящуюся рядом со слайд-шоу, которое будет воспроизводиться.

Независимо от способа запуска слайд-шоу на экране появятся изображения из слайд-шоу, которые сменяются с эффектом перекрестного затухания каждые пять секунд. Можно выбрать воспроизведение фоновой музыки. Если музыкальная запись слишком короткая по сравнению с длительностью слайд-шоу, музыка начнет воспроизводиться циклически. Чтобы выбрать портретную либо альбомную ориентацию, вращайте устройство. (При использовании эмулятора можно имитировать вращение устройства с помощью клавиш <Ctrl + F11> и <Ctrl + F12>.) Нажатие кнопки Back после завершения воспроизведения слайд-шоу или во время воспроизведения слайд-шоу возвращает вас на экран, с которого начали воспроизводить слайд-шоу.

Редактирование и удаление слайд-шоу

Чтобы отредактировать существующее слайд-шоу, коснитесь кнопки Edit Button. Можно добавить или удалить фотографии, как это делалось ранее. Выбор новой песни заменит одну из прежних. Чтобы удалить слайд-шоу из приложения, нажмите кнопку Delete.

12.3. Обзор применяемых технологий

В этом разделе будут рассмотрены новые технологии, используемые в приложении Slideshow.

Запуск объектов Intents, использующих встроенные поставщики контента

В Android не поддерживается хранилище данных, которое может использоваться всеми приложениями. Вместо этого используются поставщики (провайдеры) контента, с помощью которых приложения могут сохранять и выбирать данные, а также открывать доступ к данным для нескольких приложений. С помощью провайдеров контента в главе 9 выполнялось сохранение рисунков, созданных с помощью приложения Doodlz в Gallery устройства.

В операционную систему Android встроен ряд провайдеров контента, обеспечивающих доступ к данным (изображения, звук, видеоролики, информация о контактах и прочие данные). Просмотрите список классов в пакете android.provider, где приведен полный перечень встроенных провайдеров контентаdeveloper.android.com/reference/android/provider/package-summary.html.

В этом приложении вы можете с помощью встроенных провайдеров контента выбирать изображения либо аудиофайлы, которые находятся на устройстве и будут включены в состав слайд-шоу. Для этого запустите объекты Intent, для которых указан тип MIME данных, которые может выбирать пользователь (раздел 12.5.3). Затем Android запускает объект Activity, отображающий выбранный пользователем тип данных, либо выводит диалоговое окно выбора Activity, в котором пользователь может выбрать используемый объект Activity. Например, на рис. 12.4, *а* показан объект Activity, с помощью

которого пользователь может выбрать изображение из Gallery устройства. На рис. 12.5 показано диалоговое окно выбора объекта Activity, в котором пользователь может выбрать хранящуюся на устройстве музыку или же создать новую музыкальную запись с помощью Sound Recorder. Дополнительные сведения о провайдерах контента можно найти на веб-сайте developer.android.com/guide/topics/providers/content-providers.html.

Разработка графического интерфейса пользователя для диалогового окна AlertDialog

С помощью диалогового окна AlertDialog вы можете получить вводимые пользователем данные путем выбора его собственного компонента View для диалогового окна. Приложение Slideshow получает имя слайд-шоу от пользователя, который вводит его в компонент EditText, находящийся в диалоговом окне AlertDialog (рассматривается в разделах 12.4.6 и 12.5.2).

Настройка макета для ListActivity

При разработке приложения Address Book в главе 10 мы начали работать с компонентами ListActivity и ListView. В данном приложении мы познакомимся с заданным по умолчанию макетом ListActivity и встроенным компонентом ListView. В SlideshowEditor ListActivity используется пользовательский макет (раздел 12.4.7). После замены заданного по умолчанию макета ListActivity определите компонент ListView в макете и атрибуту android:id присвойте значение "@android:id/list".

Запуск объекта Intent, возвращающего результат

В ранее разработанных приложениях с помощью объектов Intent запускались браузеры устройств (приложение Favorite Twitter® Searches, глава 5), а также другие объекты Activity, находящиеся в том же приложении (приложение Address Book, глава 10). В обоих случаях использовался метод startActivity класса Activity, который запускал объект Activity, связанный с каждым объектом Intent. При использовании приложения Favorite Twitter® Searches пользователь мог вернуться к приложению из браузера, коснувшись кнопки Back устройства. При выполнении приложения Address Book после завершения выполнения компонента Activity пользователь автоматически возвращается к основному компоненту Activity приложения. В этом приложении впервые начал использоваться метод srartActivityForResult класса Activity, с помощью которого объект Activity извещается о завершении выполнения другого объекта Activity, а также принимает результаты выполнения от завершенного объекта Activity. Этот метод применяется для выполнения следующих операций:

- обновления компонента ListView класса Activity приложения Slideshow после изменения слайд-шоу пользователем;
- обновления компонента ListView класса Activity компонента SlideshowEditor после добавления пользователем нового изображения в слайд-шоу;
- получения местоположения изображения или отслеживания медиаклипов, добавленных пользователем в слайд-шоу.

Объект ArrayAdapter для компонента ListView

Как отмечалось в главе 10, для заполнения списков ListView используется адаптер. С помощью метода SimpleCursorAdapter выполняется ввод данных из базы данных в список ListView. В этом приложении мы расширили класс ArrayAdapter (пакет android.widget) для создания объектов, которые заполняют списки ListViews (пользовательские макеты) данными из объектов коллекций (разделы 12.5.2 и 12.5.3).

Шаблон View-Holder

Создание пользовательских элементов ListView представляет собой сложную задачу, требующую больших вычислительных ресурсов, особенно при наличии больших списков, включающих сложные макеты list-item. В процессе прокрутки списка ListView его элементы «уходят» с экрана. В Android элементы списка применяются повторно для отображения новых пунктов, которые появляются при прокрутке экрана. Повторное использование в элементах списка существующих компонентов GUI позволит увеличить производительность компонентов ListView. Чтобы реализовать это преимущество на практике, воспользуемся шаблоном view-holder. Для добавления произвольного объекта (Object) в компонент View воспользуемся методом setTag класса View. В дальнейшем доступ к данному объекту Object обеспечивается с помощью метода getTag класса View. В качестве тега объекта, который содержит (то есть включает) ссылки, используются объекты View элемента списка (то есть компоненты GUI). Использование в качестве тега компонента View обеспечивает удобный способ поддержки дополнительных сведений, которые могут использоваться в шаблоне view-holder либо в обработчиках событий (как будет демонстрироваться в этом приложении).

В процессе прокрутки нового компонента ListView в области экрана ListView проверяет наличие повторно используемого элемента списка. Если такой элемент не обнаружен, «с нуля» раздувается GUI элемента нового списка, затем сохраняются ссылки на компоненты GUI в объекте класса, который мы назовем call ViewHolder. Потом с помощью метода setTag настраивается объект ViewHolder в качестве тега для элемента ListView. Если же доступен повторно используемый элемент, получаем тег элемента с помощью метода getTag, затем возвращаем объект ViewHolder, который был ранее создан для этого элемента ListView. Независимо от того, каким образом был получен объект ViewHolder, конфигурируем различные компоненты GUI, на которые ссылается ViewHolder.

Извещение для ListView при изменении источника данных

При изменении набора данных ArrayAdapter вызывается метод notifyDataSetChanged (разделы 12.5.2 и 12.5.3), с помощью которого определяется изменение базовых данных объекта Adapter, а также обновляется соответствующий компонент ListView.

Добавление данных в компонент GUI, используемый в обработчике событий

Классы Slideshow и SlideshowEditor (разделы 12.5.2 и 12.5.3) используют методы setTag и getTag, чтобы добавить дополнительную информацию в компоненты GUI, используемую для соответствующих обработчиков событий. В класс Slideshow добавляются строки для кнопок Play и Edit, с помощью которых определяется название слайд-шоу,

которое будет воспроизводиться или редактироваться. Объект SlideshowInfo, добавляемый в кнопку Delete Button, определяет один из объектов, удаляемый из List of SlideshowInfo, который представляет все слайд-шоу.

Воспроизведение музыки с помощью MediaPlayer

Плеер MediaPlayer (пакет android.media, раздел 12.5.4) обеспечивает воспроизведение приложением аудио- или видеороликов, которые хранятся на устройстве или принимаются из сетевых потоков. С помощью MediaPlayer воспроизводятся музыкальные файлы (при их наличии), выбранные пользователем для данного слайд-шоу.

Загрузка изображений с помощью BitmapFactory

Класс BitmapFactory (пакет android.graphics) предназначен для создания объектов Bitmap. Этот класс используется в нашем приложении для загрузки изображений с устройства, используемых в качестве изображений миниатюр (разделы 12.5.2 и 12.5.3), а также для отображения во время воспроизведения слайд-шоу (раздел 12.5.4). Мы воспользовались объектом вложенного статического класса BitmapFactory.Options для конфигурирования объектов Bitmap, созданных с помощью BitmapFactory. В частности, с его помощью понижается разрешение изображений с целью экономии памяти. В результате исключается появление ошибок *out-of-memory* (за пределами памяти), которые имеют место в процессе манипулирования несколькими объектами Bitmap.

Перекрестное затухание между изображениями, реализуемое с помощью TransitionDrawable и BitmapDrawable

Во время воспроизведения слайд-шоу каждые пять секунд медленно исчезает текущее изображение и сменяется следующим изображением. Этот переход осуществляется путем отображения компонента TransitionDrawable (раздел 12.5.4), который поддерживает встроенную анимацию переходов между двумя объектами Drawable. Класс TransitionDrawable является подклассом класса Drawable и, подобно другим объектам Drawable, может отображаться с помощью ImageView. В разрабатываемом приложении изображения загружаются в виде объектов Bitmap, поэтому создаются объекты BitmapDrawables, используемые в переходах. Объекты TransitionDrawable и BitmapDrawable находятся в пакете android.graphics.drawable.

12.4. Создание графического интерфейса пользователя и файлов ресурсов

В этом разделе будут рассмотрены ресурсы и GUI-разметки приложения Slideshow. Вы уже ознакомились с созданием компонентов GUI и разметок, используемых в приложениях, а также определяли ресурсы String для каждого приложения. Поэтому в этом разделе мы не будем приводить код большинства файлов разметки либо большую часть кода файла ресурсов strings.xml. Приводятся диаграммы, отображающие названия компонентов GUI, поскольку используемые компоненты и разметки были представлены в предыдущих главах. Чтобы просмотреть содержимое файлов ресурсов и разметки, откройте их в среде Eclipse.

12.4.1. Создание проекта

Начните с создания нового проекта Android под названием Slideshow. В диалоговом окне New Android Project (Новый проект Android) укажите следующие значения, потом нажмите кнопку Finish (Готово):

- Build Target (Операционная система): Android 2.3.3.
- Application name (Имя приложения): Slideshow.
- Package name (Название пакета): com.deitel.slideshow.
- Create Activity (Создать деятельность): Slideshow.
- Min SDK Version (Минимальная версия SDK): 8.

12.4.2. Использование стандартных пиктограмм Android в графическом интерфейсе приложения

В главе 10 уже рассматривались стандартные пиктограммы, входящие в состав Android, они могут быть использованы в ваших собственных приложениях. Эти пиктограммы находятся в папке SDK's platforms внутри папки data/res/drawable-hdpi для каждой версии платформы. Некоторые из этих пиктограмм, выбранные для применения в приложении, не являются общедоступными, следовательно, могут быть недоступны для некоторых устройств Android. Исходя из этих соображений, мы скопировали используемые при разработке приложения пиктограммы в папку res/drawable-hdpi приложения. Откройте эту папку в Eclipse, чтобы просмотреть выбираемые пиктограммы.

12.4.3. Файл AndroidManifest.xml

В листинге 12.1 приведен код файла AndroidManifest.xml приложения. Этот файл включает ряд ключевых свойств, которые будут рассмотрены в этом разделе. В частности, элементы Slideshow и SlideshowEditor activity служат для отображения каждого объекта Activity в портретном режиме (строки 10 и 20). Также были выбраны темы Slideshow и SlideshowPlayer (строки 11 и 24), последняя из которых скрывает строку заголовка. В результате появляется больше места для отображения изображений, включенных в слайд-шоу.

Листинг 12.1. Файл AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android=http://schemas.android.com/apk/res/android
3     package="com.deitel.slideshow" android:versionCode="1"
4     android:versionName="1.0">
5     <application android:icon="@drawable/icon"
6        android:label="@string/app_name"
7        android:debuggable="true">
8        <activity android:name=".Slideshow"
9           android:label="@string/app_name"
10          android:screenOrientation="portrait"
11          android:theme="@android:style/Theme.Light">
12          <intent-filter>
```

продолжение ⇨

Листинг 12.1 *(продолжение)*

```
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

      <activity android:name=".SlideshowEditor"
         android:label="@string/slideshow_editor"
         android:screenOrientation="portrait"></activity>

      <activity android:name=".SlideshowPlayer"
         android:label="@string/app_name"
        android:theme="@android:style/Theme.Light.NoTitleBar"></activity>
   </application>
   <uses-sdk android:minSdkVersion="8" />
</manifest>
```

12.4.4. Разметка элементов ListView в ListActivity приложения Slideshow

На рис. 12.7 показаны диаграммы разметки элементов ListView, которые отображаются в ListActivity приложения Slideshow. Разметка, определенная в файле slideshow_list_item.xml, — это вертикальный макет LinearLayout, который содержит компонент TextView и вложенный горизонтальный макет LinearLayout. Горизонтальный макет LinearLayout содержит компонент ImageView и три кнопки Button. Каждый компонент Button использует одно новое свойство, а именно с помощью атрибута android:drawableTop отображается сообщение Drawable над текстом надписи Button. В любом случае можно использовать одну из стандартных пиктограмм Android. Например, в файле XML-разметки playButton определяется следующая ссылка: android:drawableTop="@drawable/ic_menu_play_clip", которая задает отображение картинки из файла ic_menu_play_clip.png над надписью кнопки Button. Также с помощью атрибутов android:drawableLeft, android:drawableRight и android:drawableBottom выполняется позиционирование пиктограммы слева от текста, справа от текста или под текстом соответственно.

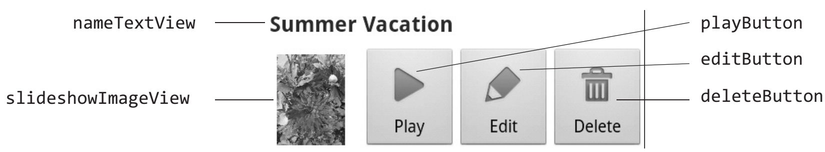


Рис. 12.7. Разметка slideshow_list_item.xml, определяющая расположение элементов ListView в ListActivity приложения Slideshow

12.4.5. Меню ListActivity приложения Slideshow

В листинге 12.2 приведена разметка для меню Slideshow из ListActivity. В качестве значков элементов меню используется стандартное изображение из файла ic_menu_slideshow.png (строка 5).

Листинг 12.2. Меню ListActivity для приложения Slideshow, определенное разметкой slideshow_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/newSlideshowItem"
      android:title="@string/menuitem_new_slideshow"
      android:icon="@drawable/ic_menu_slideshow"
      android:titleCondensed="@string/menuitem_new_slideshow"
      android:alphabeticShortcut="n"></item>
</menu>
```

12.4.6. Макет компонента EditText, определенный в диалоговом окне Set Slideshow Name

На рис. 12.8 показано диалоговое окно Set Slideshow Name, в котором пользователь может ввести название слайд-шоу в EditText. Компонент nameEditText был вложен в компонент LinearLayout, в результате чего появилась возможность настройки его левых и правых границ с помощью атрибутов android:layout_marginLeft и android:layout_marginRight соответственно. Также атрибуту android:singleLine было присвоено значение true, чтобы заголовок слайд-шоу занимал одну строку.

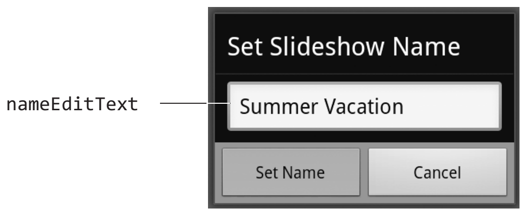


Рис. 12.8. Присваивание имени диалоговому окну AlertDialog с помощью пользовательского графического интерфейса — показано после ввода пользователем имени слайд-шоу и касания кнопки Set Name Button

12.4.7. Макет компонента ListActivity из SlideshowEditor

На рис. 12.11 показаны диаграммы макета компонента ListActivity из SlideshowEditor. Поскольку этот компонент ListActivity использует пользовательский макет (определен с помощью разметки slideshow_list_item.xml), следует определить компонент ListView в макете путем присваивания атрибуту android:id значения "@android:id/list". Этот компонент ListView возвращается с помощью метода getListView компонента ListActivity. Разметка из файла slideshow_editor.xml определяет вертикальный макет LinearLayout, который включает вложенные горизонтальные LinearLayout и ListView. Компонент LinearLayout, имеющий горизонтальный макет, включает четыре компонента Button.

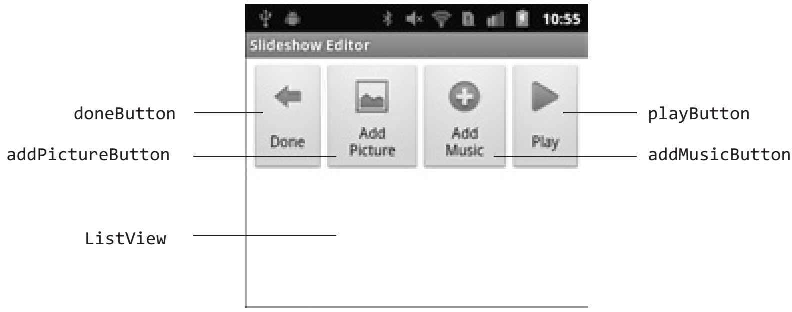


Рис. 12.9. Макет компонента ListActivity из SlideshowEditor, определенный с помощью разметки slideshow_editor.xml

12.4.8. Макет элементов ListView в SlideshowEditor

На рис. 12.10 показаны диаграммы макета для элементов ListView, которые отображаются в ListActivity из SlideshowEditor. Макет, определенный с помощью разметки slideshow_edit_item.xml, состоит из компонента с горизонтальным макетом LinearLayout, включающего компоненты ImageView и Button.

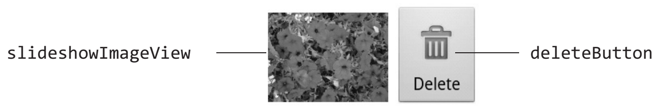


Рис. 12.10. Макет элементов ListView в ListActivity из SlideshowEditor, определенный в файле разметки slideshow_edit_item

12.4.9. Макет компонента Activity из SlideshowPlayer

На рис. 12.11 приведены диаграммы макета для объекта Activity из SlideshowPlayer. В файле разметки slideshow_edit_item.xml определен горизонтальный макет LinearLayout, содержащий компонент ImageView, который заполняет макет LinearLayout полностью.

12.5. Создание приложения

Это приложение состоит из классов SlideshowInfo (листинг 12.3), Slideshow (подкласс класса ListActivity, листинги 12.4–12.13), SlideshowEditor (подкласс класса ListActivity, листинги 12.14–12.22) и SlideshowPlayer (листинги 12.23–12.28). Главный класс Activity приложения, именуемый Slideshow, создается при создании проекта, но вы можете изменить его суперкласс на ListActivity, а потом добавить другие классы в папку проекта src/com.deitel.slideshow.



Рис. 12.11. Макет компонента ListActivity из SlideshowPlayer, определенный в файле разметки slideshow_player.xml

12.5.1. Класс SlideshowInfo

Класс SlideshowInfo (см. листинг 12.3) хранит данные для одного слайд-шоу, которые включают следующие компоненты:

- name (строка 10) — название слайд-шоу, которое отображается в списке слайд-шоу;
- imageList (строка 11) — строковый список, представляющий локации изображений;
- musicPath (строка 12) — строковое представление локации музыкальной записи (при ее наличии), которая будет воспроизводиться в фоновом режиме во время воспроизведения слайд-шоу.

Конструктор создает компонент imageList в виде массива ArrayList<String>.

Листинг 12.3. Организация хранения данных для одного слайд-шоу

```
1   // SlideshowInfo.java
2   // организация хранения данных для одного слайд-шоу.
3   package com.deitel.slideshow;
4
5   import java.util.ArrayList;
6   import java.util.List;
7
8   public class SlideshowInfo
9   {
10     private String name; // имя слайд-шоу
11     private List<String> imageList; // изоборажения этого слайд-шоу
12     private String musicPath; // локация воспроизводимой музыки
13
14     // конструктор
15     public SlideshowInfo(String slideshowName)
16     {
17        name = slideshowName; // выбор имени слайд-шоу
```

продолжение ☞

Листинг 12.3 *(продолжение)*

```
      imageList = new ArrayList<String>();
      musicPath = null; // отсутствует фоновая музыка для слайд-шоу
   } // конец определения конструктора SlideshowInfo

   // возврат имени этого слайд-шоу
   public String getName()
   {
      return name;
   } // конец определения метода getName

   // возврат списка строк, указывающих на изображения слайд-шоу
   public List<String> getImageList()
   {
      return imageList;
   } // конец определения метода getImageList

   // добавление пути к новому изображению
   public void addImage(String path)
   {
      imageList.add(path);
   } // конец определения метода addImage

   // возврат строки, соответствующей индексу позиции
   public String getImageAt(int index)
   {
      if (index >= 0 && index < imageList.size())
         return imageList.get(index);
      else
         return null;
   } // конец определения метода getImageAt

   // возврат музыки слайд-шоу
   public String getMusicPath()
   {
      return musicPath;
   } // конец определения метода getMusicPath

   // выбор музыки для слайд-шоу
   public void setMusicPath(String path)
   {
      musicPath = path;
   } // конец определения метода setMusicPath

   // возврат количества изображений/видеороликов в слайд-шоу
   public int size()
   {
      return imageList.size();
   } // конец определения метода size
} // конец определения класса SlideshowInfo
```

12.5.2. Подкласс Slideshow класса ListActivity

Класс Slideshow (см. листинги 12.4–12.12) — это главный класс Activity. Этот класс расширяет класс ListActivity, поскольку основное назначение класса Activity заключается в отображении ListView.

Операторы package, import и поля класса Slideshow

Подкласс Slideshow класса ListActivity (см. листинг 12.4) — это основной класс Activity приложения. Назначение этого класса заключается в отображении компонента ListView для всех ранее созданных слайд-шоу. Были выделены операторы import для новых классов и интерфейсов, рассмотренных в разделе 12.3 и в текущем разделе. Объекты List of SlideshowInfo (строка 41) содержат информацию обо всех созданных пользователем слайд-шоу. Этот объект List объявлен как static, поэтому может быть использован всеми другими деятельностями приложения. Класс SlideshowAdapter (строка 43) — это пользовательский класс ArrayAdapter, который отображает объекты SlideshowInfo в качестве элементов ListView.

Листинг 12.4. Операторы package, import и экземпляры переменных класса Slideshow

```
 1 // Slideshow.java
 2 // Главный класс Activity для класса Slideshow.
 3 package com.deitel.slideshow;
 4
 5 import java.util.ArrayList;
 6 import java.util.List;
 7
 8 import android.app.AlertDialog;
 9 import android.app.ListActivity;
10 import android.content.ContentResolver;
11 import android.content.Context;
12 import android.content.DialogInterface;
13 import android.content.Intent;
14 import android.graphics.Bitmap;
15 import android.graphics.BitmapFactory;
16 import android.net.Uri;
17 import android.os.AsyncTask;
18 import android.os.Bundle;
19 import android.provider.MediaStore;
20 import android.view.Gravity;
21 import android.view.LayoutInflater;
22 import android.view.Menu;
23 import android.view.MenuInflater;
24 import android.view.MenuItem;
25 import android.view.View;
26 import android.view.View.OnClickListener;
27 import android.view.ViewGroup;
28 import android.widget.ArrayAdapter;
29 import android.widget.Button;
30 import android.widget.EditText;
```

продолжение ⇗

Листинг 12.4 *(продолжение)*

```
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

public class Slideshow extends ListActivity
{
   // используется для добавления имени слайд-шоу
   // в качестве дополнительного в объект Intent
   public static final String NAME_EXTRA = "NAME";

   static List<SlideshowInfo> slideshowList; // список слайд-шоу
   private ListView slideshowListView; // ListView из ListActivity
   private SlideshowAdapter slideshowAdapter; // адаптер для ListView

```

Переопределение метода onCreate класса Activity

Метод onCreate класса Slideshow (см. листинг 12.5) получает компонент ListView, который отображает созданные пользователем слайд-шоу (строка 50), а затем создает объекты slideshowList и slideshowAdapter, а также в качестве адаптера slideshowListView выбирает slideshowAdapter. В результате slideshowListView отображает название каждого слайд-шоу, первую миниатюру, а также кнопки Button Play, Edit и Delete. При этом используется разметка, определенная в файле slideshow_list_item.xml (раздел 12.4.4). В строках 58–62 создается и отображается диалоговое окно AlertDialog, в котором содержится информация для пользователя о запуске приложения.

Листинг 12.5. Переопределение метода onCreate класса Activity

```
45   // вызывается при первом создании деятельности
46   @Override
47   public void onCreate(Bundle savedInstanceState)
48   {
49      super.onCreate(savedInstanceState);
50      slideshowListView = getListView(); // получаем встроенный ListView
51
52      // создание и настройка адаптера ListView
53      slideshowList = new ArrayList<SlideshowInfo>();
54      slideshowAdapter = new SlideshowAdapter(this, slideshowList);
55      slideshowListView.setAdapter(slideshowAdapter);
56
57     // создание построителя (Buider) для нового диалогового
       // окна AlertDialog
58     AlertDialog.Builder builder = new AlertDialog.Builder(this);
59     builder.setTitle(R.string.welcome_message_title);
60     builder.setMessage(R.string.welcome_message);
61     builder.setPositiveButton(R.string.button_ok, null);
62     builder.show();
63  }  // конец определения метода onCreate
64
```

Переопределение методов onCreateOptionsMenu, onOptionsItemSelected и onActivityResult класса Activity

Метод onCreateOptionsMenu (см. листинг 12.6, строки 66–73) «раздувает» меню класса Activity из файла разметки slideshow_menu.xml (раздел 12.4.5). После выбора пользователем элемента меню New Slideshow метод onOptionsItemSelected (строки 79–132) отображает диалоговое окно с пользовательским интерфейсом, с помощью которого пользователь вводит название слайд-шоу. Для отображения компонента EditText в диалоговом окне мы «раздуваем» разметку slideshow_name_edittext.xml (строка 87) и выбираем ее в качестве компонента View для диалогового окна (строка 93). После выбора пользователем кнопки OK в диалоговом окне метод onClick (строки 99–124) получает имя от компонента EditText, создает новый объект SlideshowInfo для слайд-шоу и добавляет его в список slideshowList. В строках 110–112 конфигурируется объект Intent, применяемый для запуска Activity из SlideshowEditor. Затем в строке 113 запускается объект Intent с помощью метода startActivityForResult. Первый аргумент для этого метода является объектом Intent, представляющим запускаемый объект суб-Activity. Второй аргумент — это положительный код запроса, который идентифицирует объект Activity, возвращающий результат. Это значение принимается в качестве первого параметра метода onActivityResult (строки 135–141), который вызывается в случае, если возвращается суб-Activity. В результате этот объект Activity может обрабатывать результат. Если объект Activity может запускать несколько других объектов, код запроса применяется в методе onActivityResult для идентификации возвращаемого суб-Activity, в результате чего выполняется корректная обработка результата. С момента запуска единственного суб-Activity из данного Activity мы использовали значение 0 (определено как константа EDIT_ID в строке 76) для второго аргумента. Используя отрицательный результат, код заставляет startActivityForResult работать так же, как и startActivity. Если система не может найти Activity для обработки этого объекта Intent, метод startActivityForResult генерирует исключение ActivityNotFoundException.

ПРИМЕЧАНИЕ

В общем случае следует заключить вызовы startActivity и startActivityForResult в операторы try, что позволит вызывать исключение в случае, если отсутствует Activity, обрабатывающий Intent.

Листинг 12.6. Переопределение методов onCreateOptionsMenu, onOptionsItemSelected и onActivityResult класса Activity

```
65   // создание меню Activity на основе XML-файла ресурсов меню
66   @Override
67   public boolean onCreateOptionsMenu(Menu menu)
68   {
69      super.onCreateOptionsMenu(menu);
70      MenuInflater inflater = getMenuInflater();
71      inflater.inflate(R.menu.slideshow_menu, menu);
72      return true;
73   }  // конец определения метода onCreateOptionsMenu
74
```

продолжение ☞

Листинг 12.6 *(продолжение)*

```
// код запроса SlideshowEditor передается методу startActivityForResult
private static final int EDIT_ID = 0;

// обработка варианта, выбранного в меню параметров
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
   // получение ссылки на службу LayoutInflater
   LayoutInflater inflater = (LayoutInflater) getSystemService(
      Context.LAYOUT_INFLATER_SERVICE);

   // «рздувание» slideshow_name_edittext.xml для создания EditText
   View view = inflater.inflate(R.layout.slideshow_name_edittext, null);
   final EditText nameEditText =
      (EditText) view.findViewById(R.id.nameEditText);

   // создание диалогового окна ввода для именования слайд-шоу
   AlertDialog.Builder inputDialog = new AlertDialog.Builder(this);
   inputDialog.setView(view); // настройка пользовательского
                              // просмотра диалогового окна
   inputDialog.setTitle(R.string.dialog_set_name_title);

   inputDialog.setPositiveButton(R.string.button_set_slideshow_name,
      new DialogInterface.OnClickListener()
   {
      public void onClick(DialogInterface dialog, int whichButton)
      {
         // создание класса SlideshowInfo для нового слайд-шоу
         String name = nameEditText.getText().toString().trim();

         if (name.length() != 0)
         {
            slideshowList.add(new SlideshowInfo(name));

            // создание Intent для запуска SlideshowEditor Activity,
            // добавление имени слайд-шоу в качестве дополнительного
            // и запуск Activity
            Intent editSlideshowIntent =
               new Intent(Slideshow.this, SlideshowEditor.class);
            editSlideshowIntent.putExtra("NAME_EXTRA", name);
            startActivityForResult(editSlideshowIntent, 0);
         }  // конец блока if
            else
            {
               // отображение сообщения о необходимости
               // именования слайд-шоу
               Toast message = Toast.makeText(Slideshow.this,
                  R.string.message_name, Toast.LENGTH_SHORT);
               message.setGravity(Gravity.CENTER,
                  message.getXOffset() / 2, message.getYOffset() / 2);
```

```
122                message.show(); // отображение сообщения Toast
123             }  // конец блока else
124          }  // конец определения метода onClick
125       } // конец определения анонимного внутреннего класса
126    );// завершение вызова setPositiveButton
127
128    inputDialog.setNegativeButton(R.string.button_cancel, null);
129    inputDialog.show();
130
131  return super.onOptionsItemSelected(item); // вызов метода суперкласса
132 } // конец определения метода onOptionsItemSelected
133
134 // обновление ListView после завершения редактирования слайд-шоу
135 @Override
136 protected void onActivityResult(int requestCode, int resultCode,
137    Intent data)
138 {
139    super.onActivityResult(requestCode, resultCode, data);
140    slideshowAdapter.notifyDataSetChanged(); // обновление адаптера
141 }  // конец описания метода onActivityResult
142
```

Переопределенный метод onActivityResult класса Activity (строки 135–141) вызывается в случае, если другой класс Activity возвращает для него результат. Параметр requestCode — это значение, передаваемое в виде второго параметра методу startActivityForResult в случае запуска другого класса Activity. Значение параметра resultCode будет следующим:

- RESULT_OK, если выполнение Activity успешно завершено;
- RESULT_CANCELED в случае, если Activity не возвращает результат, либо его выполнение завершилось крахом, либо если Activity явно вызывает метод setResult с аргументом RESULT_CANCELED.

Третий параметр — это объект Intent, содержащий данные (в форме дополнений), возвращаемый данному классу Activity. В рассматриваемом примере требуется просто узнать, был ли завершен данный Activity из SlideshowEditor, в результате чего можно обновить ListView новым слайд-шоу. С помощью метода notifyDataSetChanged класса SlideshowAdapter формируется уведомление относительно изменения базовых данных адаптера и обновления компонента ListView.

Объект SlideshowAdapter: использование шаблона View-Holder для заполнения ListView

В листинге 12.7 определяются частные вложенные классы ViewHolder и SlideshowAdapter. Класс ViewHolder просто определяет переменные экземпляра класса package-access, в результате чего класс SlideshowAdapter получает возможность непосредственного доступа при манипулировании объектами ViewHolder. Если создан элемент ListView, создаем объект класса ViewHolder и связываем его с этим элементом ListView. Если существующий компонент ListView был повторно использован, мы просто получаем доступ к объекту ViewHolder, который был ранее связан с этим компонентом.

Листинг 12.7. Класс SlideshowAdapter, предназначенный для заполнения данными компонента ListView

```
// класс, реализующий шаблон "ViewHolder pattern"
// для улучшения производительности ListView
private static class ViewHolder
{
   TextView nameTextView; // ссылка на TextView элемента ListView
   ImageView imageView;   // ссылка на ImageView элемента ListView
   Button playButton;     // ссылка на кнопку Play элемента ListView
   Button editButton;     // ссылка на кнопку Edit элемента ListView
   Button deleteButton;   // ссылка на кнопку Delete элемента ListView
}  // конец определения класса ViewHolder

// Подкласс ArrayAdapter, отображающий имя слайд-шоу, первое
// изображение и кнопки "Play", "Edit" и "Delete"
private class SlideshowAdapter extends ArrayAdapter<SlideshowInfo>
{
   private List<SlideshowInfo> items;
   private LayoutInflater inflater;

   // общедоступный конструктор для SlideshowAdapter
   public SlideshowAdapter(Context context, List<SlideshowInfo> items)
   {
      // вызов конструктора суперкласса
      super(context, -1, items);
      this.items = items;
      inflater = (LayoutInflater)
         getSystemService(Context.LAYOUT_INFLATER_SERVICE);
   }  // конец описания конструктора SlideshowAdapter

   // возвращение компонента View для отображения в заданной позиции
   @Override
   public View getView(int position, View convertView,
      ViewGroup parent)
   {
      ViewHolder viewHolder; // ссылки на GUI текущего элемента

      // если convertView равно null, "раздувание" GUI
      // и создание ViewHolder либо получение существующего ViewHolder
      if (convertView == null)
         {
            convertView =
               inflater.inflate(R.layout.slideshow_list_item, null);

            // настройка ViewHolder для этого элемента ListView
            viewHolder = new ViewHolder();
            viewHolder.nameTextView = (TextView)
               convertView.findViewById(R.id.nameTextView);
               viewHolder.imageView = (ImageView)
            convertView.findViewById(R.id.slideshowImageView);
```

```
               viewHolder.playButton =
                  (Button) convertView.findViewById(R.id.playButton);
               viewHolder.editButton =
                  (Button) convertView.findViewById(R.id.editButton);
               viewHolder.deleteButton =
                  (Button) convertView.findViewById(R.id.deleteButton);
               convertView.setTag(viewHolder); // сохранить тег View
            }  // конец блока if
            else // получить ViewHolder из тега convertView
               viewHolder = (ViewHolder) convertView.getTag();

            // получить имя слайд-шоу, отображаемого в nameTextView
            SlideshowInfo slideshowInfo = items.get(position);
            viewHolder.nameTextView.setText(slideshowInfo.getName());

            // есди в данном слайд-шоу есть хоть одно изображение
            if (slideshowInfo.size() > a)
            {
               // создание растра на основе первого изображения
               // или видеоролика слайд-шоу
               String firstItem = slideshowInfo.getImageAt(0);
               new LoadThumbnailTask().execute(viewHolder.imageView,
                  Uri.parse(firstItem));
            }  // конец блока if

            // настройка тега и OnClickListener для кнопки "Play"
            viewHolder.playButton.setTag(slideshowInfo);
            viewHolder.playButton.setOnClickListener(playButtonListener);

            // создание и установка OnClickListener для кнопки "Edit"
            viewHolder.editButton.setTag(slideshowInfo);
            viewHolder.editButton.setOnClickListener(editButtonListener);

            // создание и настройка OnClickListener для кнопки "Delete"
            viewHolder.deleteButton.setTag(slideshowInfo);
        viewHolder.deleteButton.setOnClickListener(deleteButtonListener);

            return convertView; // возврат View в этой позиции
         }  // конец определения getView
      }  // конец определения класса SlideshowAdapter

```

В приложении AddressBook был создан класс SimpleCursorAdapter, предназначенный для отображения строк String (имен контактов), хранящихся в базе данных. Кроме того, был создан адаптер, который специально предназначался для отображения строк String и изображений на компоненты TextView и ImageView соответственно. Компоненты ListView, используемые в этом приложении, являются более сложными. Каждый из них содержит текст (имя слайд-шоу), изображение (первая картинка из слайд-шоу) и кнопки Button (Play, Edit и Delete). Чтобы отобразить данные слайд-шоу на эти компоненты ListView, мы расширили класс ArrayAdapter таким образом, чтобы можно было переопределить метод getView, применяемый для конфигурирования пользовательского

макета для каждого компонента ListView. Конструктор (строки 162–169) вызывает конструктор суперкласса, потом сохраняет объекты List of SlideshowInfo и LayoutInflater для использования в дальнейшем с помощью метода getView. Второй аргумент конструктора суперкласса представляет ID ресурса макета, который включает TextView, применяемый для отображения данных в элементе ListView. В этом случае мы настраиваем его позже, поэтому в качестве значения данного аргумента указываем -1.

Метод getView (строки 172–228) выполняет пользовательское отображение данных на компонент ListView. Этот метод в качестве аргументов принимает позицию компонента ListView, компонент View (convertView), представляющий элемент ListView, и родительский компонент ListView. Путем манипулирования содержимым convertView можно настроить контент элемента ListView. Если значение convertView равно null, в строках 182–196 «раздувается» разметка slideshow_list_item.xml элемента ListView, которая присваивается convertView. Потом создается объект ViewHolder и присваиваются компоненты GUI, которые были «раздуты» в переменные экземпляра класса ViewHolder. В строке 197 этот объект ViewHolder настраивается как тег элемента ListView. Если convertView не равен null, компонент ListView повторно использует компонент ListView, который при прокрутке «уходит» за пределы экрана. В этом случае в строке 200 получаем тег элемента ListView, а также повторно используется данный объект ViewHolder. В строке 203 получаем объект SlideshowInfo, который соответствует позиции элемента ListView.

В строке 204 полю nameTextView объекта viewHolder присваивается имя слайд-шоу. Если слайд-шоу включает изображения, в строках 210–212 определяется путь к первому изображению, а потом создается новый объект LoadThumbnailTask AsyncTask (листинг 12.8), предназначенный для загрузки и отображения миниатюры изображения на imageView объекта viewHolder.

В строках 216–225 настраиваются слушатели для кнопок Play, Edit и Delete данного элемента ListView. В каждом случае используется метод setTag класса Button для поддержки некоторой дополнительной информации (в форме Object), которая нужна соответствующему обработчику событий (например, в форме объекта SlideshowInfo, представляющего слайд-шоу). Для обработчиков событий playButton и editButton этот объект используется в качестве дополнения к объекту Intent, в результате чего SlideshowPlayer и SlideshowEditor «знают», какое слайд-шоу нужно воспроизводить или изменять соответственно. Для кнопки deleteButton поддерживается объект SlideshowInfo, который может быть удален из объектов списка List of SlideshowInfo.

Вложенный класс LoadThumbnailTask

Класс LoadThumbnailTask (см. листинг 12.8) загружает миниатюру изображения в отдельный поток выполнения, что позволяет гарантировать требуемую степень «отзывчивости» потока GUI. Метод doInBackground используется метод утилиты getThumbnail типа static для загрузки миниатюры. После завершения загрузки метод onPostExecute получает Bitmap миниатюры и отображает его в отдельном компоненте ImageView.

Листинг 12.8. Класс LoadThumbnailTask загружает миниатюру в отдельный поток

```
   // задача по загрузке миниатюры в отдельный поток
  private class LoadThumbnailTask extends AsyncTask<Object,Object,Bitmap>
   {
```

```
234      ImageView imageView; // отображает миниатюру
235
236      // загрузка миниатюры: в качестве аргументов ImageView и Uri
237      @Override
238      protected Bitmap doInBackground(Object... params)
239      {
240         imageView = (ImageView) params[0];
241
242         return Slideshow.getThumbnail((Uri) params[1],
243            getContentResolver(), new BitmapFactory.Options());
244      }  // конец определения метода doInBackground
245
246      // установка миниатюры для ListView
247      @Override
248      protected void onPostExecute(Bitmap result)
249      {
250         super.onPostExecute(result);
251         imageView.setImageBitmap(result);
252      }  // конец определения метода onPostExecute
253   }  // конец определения класса LoadThumbnailTask
254
```

Слушатель playButtonListener из OnClickListener, реагирующий на события объекта playButton выбранного слайд-шоу

Слушатель playButtonLIstener из OnClickListener (см. листинг 12.9) обрабатывает события объекта playButton. Мы создали обьект Intent, предназначенный для запуска Activity из SlideshowPlayer и добавления имени слайд-шоу в качестве дополнения Intent (строки 262–265). Аргументы — это строковые данные, которые тегируют дополнительные данные и тегированные значения (название слайд-шоу). В строке 265 используется метод getTag класса View для получения значения, которое устанавливается с помощью метода setTag (название слайд-шоу) в строке 216. В строке 266 запускается объект Intent.

Листинг 12.9. Слушатель событий, обрабатывающий событие click для метода playButton

```
255   // обработка событий, сгенерированных кнопкой "Play"
256   OnClickListener playButtonListener = new OnClickListener()
257   {
258      @Override
259      public void onClick(View v)
260      {
261         // создание объекта intent для запуска Activity SlideshowPlayer
262         Intent playSlideshow =
263            new Intent(Slideshow.this, SlideshowPlayer.class);
264         playSlideshow.putExtra(
265            NAME_EXTRA, ((SlideshowInfo) v.getTag()).getName());
266         startActivity(playSlideshow); // launch SlideshowPlayer Activity
267      }  // конец определения метода onClick
268   }; // конец определения playButtonListener
269
```

Слушатель editButtonListener из OnClickListener, реагирующий на события editButton для выбранного слайд-шоу

Слушатель editButtonLIstener из OnClickListener (см. листинг 12.10) обрабатывает события объекта editButton. Сначала создается объект Intent, предназначенный для запуска Activity из SlideshowEditor, затем добавляется имя слайд-шоу в виде дополнения объекта Intent (строки 277–280). В строке 280 используется метод getTag класса View для получения значения, которое может быть настроено с помощью метода setTag (то есть имя слайд-шоу) в строке 220. В строке 281 запускается объект Intent с помощью метода startActivityForResult. Компонент ListView из данного Activity может быть обновлен с помощью onActivityResult (в случае если пользователь изменяет первое изображение слайд-шоу во время редактирования).

Листинг 12.10. Слушатель событий для события click объекта editButton

```
270   // обработка событий, сгенерированных кнопкой "Edit"
271   private OnClickListener editButtonListener = new OnClickListener()
272   {
273      @Override
274      public void onClick(View v)
275      {
276         // создание объекта intent, предназначенного для запуска
            // Activity из SlideshowEditor
277         Intent editSlideshow =
278            new Intent(Slideshow.this, SlideshowEditor.class);
279         editSlideshow.putExtra(
280            NAME_EXTRA, ((SlideshowInfo) v.getTag()).getName());
281         startActivityForResult(editSlideshow, 0);
282      }  // конец описания метода onClick
283   }; // конец описания слушателя playButtonListener
284
```

Слушатель deleteButtonListener из OnClickListener, реагирующий на события deleteButton для выбранного слайд-шоу

Слушатель deleteButtonLIstener из OnClickListener (см. листинг 12.11) реагирует на события, генерируемые deleteButton. При удалении слайд-шоу от пользователя требуется подтверждение этой операции. Если удаление подтверждается, с помощью метода getTag класса View обеспечивается доступ к объекту SlideshowInfo, который был установлен с помощью метода setTag в строке 224. Затем удаляется объект из списка slideshowList. В строке 304 обновляется компонент ListView путем вызова метода notifyDataSetChanged класса slideshowAdapter.

Листинг 12.11. Слушатель событий, предназначенный для обработки события щелчка объекта deleteButton

```
285   // обработка событий, сгенерированных кнопкой "Delete"
286   private OnClickListener deleteButtonListener = new OnClickListener()
287   {
288      @Override
```

```
    public void onClick(final View v)
    {
       // создание построителя нового диалогового окна AlertDialog
       AlertDialog.Builder builder =
          new AlertDialog.Builder(Slideshow.this);
          builder.setTitle(R.string.dialog_confirm_delete);
          builder.setMessage(R.string.dialog_confirm_delete_message);
          builder.setPositiveButton(R.string.button_ok,
             new DialogInterface.OnClickListener()
             {
                @Override
                public void onClick(DialogInterface dialog, int which)
                {
                   Slideshow.slideshowList.remove(
                      (SlideshowInfo) v.getTag());
                   slideshowAdapter.notifyDataSetChanged();
                   //обновление
                } // конец определения метода onClick
             } // конец определения анонимного внутреннего класса
       ); // конец вызова метода setPositiveButton
       builder.setNegativeButton(R.string.button_cancel, null);
       builder.show();
    } // конец определения метода onClick
 }; // конец определения слушателя playButtonListener

```

Метод getSlideshowInfo

В листинге 12.12 определяется метод утилиты getSlideshowInfo, возвращающий выбранный объект SlideshowInfo. Этот метод выполняет итерации по объектам List of SlideshowInfo и сравнивает имя с именем, сохраненным в каждом объекте. Если соответствующий объект SlideshowInfo найден, в строке 319 этот объект возвращается. Если объект не найден, в строке 321 возвращается значение null.

Листинг 12.12. Метод утилиты getSlideshowInfo, возвращающий объект SlideshowInfo для слайд-шоу с указанным именем

```
313 // метод утилиты, предназначенный для поиска объекта SlideshowInfo
    // по имени слайд-шоу
314 public static SlideshowInfo getSlideshowInfo(String name)
315 {
316    // для каждого объекта SlideshowInfo
317    for (SlideshowInfo slideshowInfo : slideshowList)
318       if (slideshowInfo.getName().equals(name))
319          return slideshowInfo;
320
321    return null; // отсутствует соответствующий объект
322 } // конец определения метода getSlideshowInfo
323
```

Метод getThumbnail

В листинге 12.13 определяется метод утилиты getThumbnail, который принимает три аргумента, — Uri, представляющий местонахождение изображения, ContentResolver, предназначенный для взаимодействия с файловой системой устройства, и объект BitmapFactory.Options, определяющий конфигурацию Bitmap. В строке 328 из аргумента Uri извлекается id изображения, для которого будет загружена миниатюра. В строках 330–331 с помощью Android MediaStore получается соответствующее изображение миниатюры. Класс MediaStore.Images.Thumbnails поддерживает собственный метод утилиты getThumbnail, предназначенный для этих же целей. В качестве аргументов этого метода используется ContentResolver, предназначенный для взаимодействия с файловой системой устройства, id изображения, тип загружаемой миниатюры и BitmapFactory. С помощью Options определяется конфигурация Bitmap. В строке 333 возвращается объект Bitmap.

Листинг 12.13. Метод утилиты getThumbnail загружает Bitmap миниатюры по указанному Uri

```
324   //метод утилиты, предназначенный для загрузки Bitmap
      // миниатюры изображения
325   public static Bitmap getThumbnail(Uri uri, ContentResolver cr,
326      BitmapFactory.Options options)
327   {
328      int id = Integer.parseInt(uri.getLastPathSegment());
329
330      Bitmap bitmap = MediaStore.Images.Thumbnails.getThumbnail(cr, id,
331         MediaStore.Images.Thumbnails.MICRO_KIND, options);
332
333         return bitmap;
334   } // конец определения метода getThumbnail
335 } // конец определения класса Slideshow
```

12.5.3. Подкласс SlideshowEditor класса ListActivity

С помощью класса SlideshowEditor (см. листинги 12.14–12.22) пользователи могут добавлять изображения на фон, а также включать фоновые аудиоклипы в слайд-шоу. Этот класс расширяет класс ListActivity, поскольку основное назначение этого класса Activity заключается в отображении компонентов ListView для изображений в слайд-шоу. Как упоминалось в разделе 12.4.7, компонент ListActivity использует пользовательский макет.

Операторы package, import и переменные экземпляра класса SlideshowEditor

В листинге 12.14 начинается определение класса SlideShowEditor. Мы выделили операторы import для новых классов и интерфейсов, рассмотренных в разделе 12.3 и в этом разделе. Класс SlideshowEditorAdapter (строка 26) — это пользовательский подкласс класса ArrayAdapter, применяемый для отображения изображений слайд-шоу, которые были изменены с помощью данного компонента ListView класса Activity. Каждое изображение в слайд-шоу отображается в виде элемента ListView, показывающего кнопку Delete Button

(Удалить), которая применяется для удаления изображения слайд-шоу. Измененное слайд-шоу представлено объектом SlideshowInfo, который объявляется в строке 27.

Листинг 12.14. Операторы package, import и переменные экземпляра класса SlideshowEditor

```
1  // SlideshowEditor.java
2  // Класс Activity, применяемый для создания и изменения слайд-шоу.
3  package com.deitel.slideshow;
4
5  import java.util.List;
6
7  import android.app.ListActivity;
8  import android.content.Context;
9  import android.content.Intent;
10 import android.graphics.Bitmap;
11 import android.graphics.BitmapFactory;
12 import android.net.Uri;
13 import android.os.AsyncTask;
14 import android.os.Bundle;
15 import android.view.LayoutInflater;
16 import android.view.View;
17 import android.view.View.OnClickListener;
18 import android.view.ViewGroup;
19 import android.widget.ArrayAdapter;
20 import android.widget.Button;
21 import android.widget.ImageView;
22
23 public class SlideshowEditor extends ListActivity
24 {
25    // класс slideshowEditorAdapter, отображающий слайд-шоу в ListView
26    private SlideshowEditorAdapter slideshowEditorAdapter;
27    private SlideshowInfo slideshow; // данные слайд-шоу
28
```

Переопределение метода onCreate класса Activity

В листинге 12.15 переопределяется метод onCreate, конфигурирующий пользовательский интерфейс Activity. В строке 34 в качестве макета ListActivity выбирается разметка, определенная в файле slideshow_editor.xml. В строке 37 формируется объект Intent, который запускает этот класс Activity, затем получает дополнение String под названием Slideshow.NAME_EXTRA, которое хранится в Bundle из Intent. В строке 38 с помощью статического метода getSlideshowInfo класса Slideshow (см. листинг 12.12) получаем объект SlideshowInfo для слайд-шоу, которое было создано первый раз или изменено. В строках 41–52 получаем ссылки на кнопки Button в GUI и регистрируем обработчики событий этих кнопок. В строках 55–56 создается новый класс SlideshowEditorAdapter (см. листинг 12.22), применяемый для отображения каждого элемента слайд-шоу с помощью разметки list-item, определенной в файле slideshow_edit_item.xml. Затем в качестве адаптера ListView выбирается SlideshowEditorAdapter.

Листинг 12.15. Переопределение метода onCreate класса Activity

```
// вызывается после создания деятельности
@Override
public void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.slideshow_editor);

   // выборка слайд-шоу
   String name = getIntent().getStringExtra(Slideshow.NAME_EXTRA);
   slideshow = Slideshow.getSlideshowInfo(name);

   // настройка обработчиков событий OnClickListeners для каждой
   // кнопки Button
   Button doneButton = (Button) findViewById(R.id.doneButton);
   doneButton.setOnClickListener(doneButtonListener);

   Button addPictureButton =
      (Button) findViewById(R.id.addPictureButton);
   addPictureButton.setOnClickListener(addPictureButtonListener);

   Button addMusicButton = (Button) findViewById(R.id.addMusicButton);
   addMusicButton.setOnClickListener(addMusicButtonListener);

   Button playButton = (Button) findViewById(R.id.playButton);
   playButton.setOnClickListener(playButtonListener);

   // получение ListView и настройка его адаптера для отображения
   // списка отображений
   SlideshowEditorAdapter =
      new SlideshowEditorAdapter(this, slideshow.getImageList());
   getListView().setAdapter(slideshowEditorAdapter);
} // конец определения метода onCreate

```

Переопределение метода onActivityResult класса Activity

Как отмечалось в разделе 12.5.2, метод onActivityResult (см. листинг 12.16) вызывается в случае, если класс subActivity, запущенный с помощью метода startActivityForResult, завершает свое выполнение. Немного позже вы увидите, что SlideshowEditor запускает один класс Activity, с помощью которого пользователь может выбирать изображение, хранящееся на устройстве, а также другой класс Activity, позволяющий выбирать музыку. И поскольку запущено несколько классов subActivity, мы используем константы в строках 61–62, определяющих коды запроса, с помощью которых определяется subActivity, возвращающий результаты объекту onActivityResult. Код запроса применяется для запуска класса Activity с startActivityForResult, передаваемым методу onActivityResult в качестве первого аргумента. Параметр resultCode принимает значение RESULT_OK (строка 69) в случае, если возвращающий его класс Activity был выполнен успешно. Обработка результата выполняется только в том случае, если отсутствуют

ошибки. Параметру Intent передается результат выполнения класса Activity. В строке 71 используется метод getData класса Intent для получения параметра Uri, представляющего изображение или музыку, которые выбирает пользователь. Если вызывается метод onActivityResult после выбора изображения (строка 74), в строке 77 добавляется путь к изображениям в список путей к изображениям в слайд-шоу. В строке 80 указывается, что набор данных SlideshowEditorAdapter был изменен, в результате чего компонент ListView из SlideshowEditor может быть изменен. Если после выбора музыкального сопровождения вызывается метод onActivityResult (строка 82), в строке 83 устанавливается путь к музыкальным файлам слайд-шоу.

Листинг 12.16. Переопределение метода onActivityResult класса Activity

```
60   // настройка идентификаторов ID для каждого типа
     // результирующего медиаресурса
61   private static final int PICTURE_ID = 1;
62   private static final int MUSIC_ID = 2;
63
64   // вызывается, если возвращается класс Activity, запущенный текущим
     // классом Activity
65   @Override
66   protected void onActivityResult(int requestCode, int resultCode,
67      Intent data)
68   {
69      if (resultCode == RESULT_OK) // если ошибки отсутствуют
70         {
71            Uri selectedUri = data.getData();
72
73            // если Activity возвращает изображение
74            if (requestCode == PICTURE_ID)
75            {
76               // добавление нового пути к изображению в слайд-шоу
77               slideshow.addImage(selectedUri.toString());
78
79               // обновление содержимого списка ListView
80               slideshowEditorAdapter.notifyDataSetChanged();
81            }  // end if
82            else if (requestCode == MUSIC_ID) // Activity
                                             // возвращает музыку
83               slideshow.setMusicPath(selectedUri.toString());
84         }  // конец блока if
85    }  // конец определения метода onActivityResult
86
```

Слушатель doneButtonListener метода OnClickListener, обрабатывающий событие щелчка doneButton

После касания пальцем кнопки doneButton слушатель doneButtonListener (см. листинг 12.17) вызывает метод finish класса Activity (строка 94), производящий завершение выполнения класса Activity и возврат к вызывающему классу.

Листинг 12.17. Слушатель backButtonListener метода OnClickListener обрабатывает события кнопки backButton

```
// Вызывается после выбора пользователем кнопки "Done"
private OnClickListener doneButtonListener = new OnClickListener()
{
   // возврат к предыдущему объекту Activity
   @Override
   public void onClick(View v)
   {
      finish();
   } // конец определения метода onClick
}; // конец определения слушателя doneButtonListener из OnClickListener

```

Слушатель addPictureButtonListener из метода OnClickListener, предназначенный для обработки событий щелчков для addPictureButton

Слушатель addPictureButtonListener (см. листинг 12.18) запускает внешний объект Activity, осуществляющий выбор изображений (например, Gallery) после щелчка на кнопке addPictureButton. В строке 105 создается новый объект Intent, включающий константу ACTION_GET_CONTENT, которая определяет возможность пользователя выбирать содержимое объекта Intent, хранящееся на устройстве. Методу setType класса Intent передается строка String, которая представляет тип MIME изображения, выключая возможность выбора изображения пользователем. Значок звездочки (*) в определении типа MIME определяет возможность выбора изображения любого типа. Метод createChooser класса Intent возвращает выбранный объект Intent в качестве одного из типов android.intent.action.CHOOSER. При этом обращается окно выбора Activity, в котором пользователь может выбрать объект Activity, используемый для выбора изображения (если устройство поддерживает несколько типов объектов Activity). Если поддерживается единственный объект Activity, выполняется его запуск на выполнение. Например, на используемом нами тестовом устройстве можно выбирать изображения только из приложения Gallery. Второй аргумент метода createChooser — это заголовок, который отображается в окне выбора Activity.

Листинг 12.18. Слушатель метода addPictureButtonListener метода OnClickListener обрабатывает события кнопки addPictureButton

```
 98 // вызывается после выбора пользователем кнопки "Add Picture" Button
 99 private OnClickListener addPictureButtonListener =
       new OnClickListener()
100 {
101    // запуск деятельности по выбору изображения
102    @Override
103    public void onClick(View v)
104    {
105       Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
106       intent.setType("image/*");
```

```
         startActivityForResult(Intent.createChooser(intent,
            getResources().getText(R.string.chooser_music)), MUSIC_ID);
      }  // конец определения метода onClick
  }; // конец определения слушателя addPictureButtonListener
      // из метода OnClickListener

```

Слушатель addMusicButtonListener из метода OnClickListener, обрабатывающий события щелчка для объекта addMusicButton

Слушатель addMusicButtonListener метода OnClickListener (см. листинг 12.19) запускает внешний объект Activity, выполняющий выбор музыки, с помощью которого выбирается фоновая музыка для слайд-шоу. Этот обработчик событий работает подобно обработчику событий, код которого приведен в листинге 12.18, за исключением того, что объект Intent использует MIME-тип "audio/*", обеспечивающий выбор пользователем аудиофайла произвольного типа на устройстве. На типичном устройстве, запускающем этот объект Intent, отображается окно выбора, показанное на рис. 12.12. В этом окне пользователь может выбрать музыкальные записи (с помощью кнопки Select music track) либо записать новый аудиоклип (с помощью кнопки Sound Recorder).

Листинг 12.19. Слушатель addMusicButtonListener метода OnClickListener обрабатывает события addMusicButton листинга

```
// вызывается после выбора пользователем кнопки "Add Music" Button
private OnClickListener addMusicButtonListener = new OnClickListener()
{
   // запуск деятельности по выбору музыки
   @Override
   public void onClick(View v)
   {
      Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
      intent.setType("audio/*");
      startActivityForResult(Intent.createChooser(intent,
         getResources().getText(R.string.chooser_music)), MUSIC_ID);
   } // завершение описания метода onClick
}; // завершение описания слушателя addMusicButtonListener
   // из метода OnClickListener

```

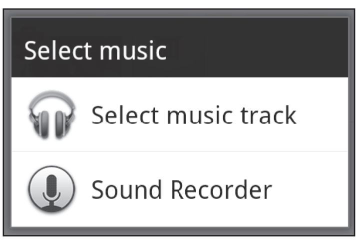


Рис. 12.12. В этом окне можно выбрать музыкальную запись или записать новый аудиоклип

Слушатель playButtonListener метода OnClickListener, предназначенный для обработки событий щелчка кнопки PlayButton

Слушатель playButtonListener метода OnClickListener (см. листинг 12.20) запускает объект Activity класса SlideshowPlayer после выбора пользователем кнопки Play Button. В строках 137–142 создается новый объект Intent для класса SlideshowPlayer, который включает название слайд-шоу в качестве дополнения Intent, а потом запускает объект Intent.

Листинг 12.20. Слушатель playButtonListener метода OnClickListener, обрабатывающий события кнопки playButton

```
126 // вызывается, если пользователь нажал кнопку "Play" Button
127 private OnClickListener playButtonListener = new OnClickListener()
128 {
129    // воспроизведение текущего слайд-шоу
130    @Override
131    public void onClick(View v)
132    {
133       // создание нового Intent для запуска SlideshowPlayer Activity
134       Intent playSlideshow =
135          new Intent(SlideshowEditor.this, SlideshowPlayer.class);
136
137       // включение имени слайд-шоу в качестве дополнения
138       playSlideshow.putExtra(
139       Slideshow.NAME_EXTRA, slideshow.getName());
140       startActivity(playSlideshow); // запуск Activity
141    }  // конец описания метода onClick
142 };  // конец описания playButtonListener
143
```

Слушатель deleteButtonListener метода OnClickListener, применяемый для обработки событий щелчка кнопки deleteButton

Слушатель OnClickListener из deleteImage (см. листинг 12.21) удаляет изображение с помощью соответствующей кнопки Delete Button, которая была активирована. Кнопка Delete сохраняет путь к связанному изображению в виде его тега. В строке 152 вызывается тег, который передается методу remove класса slideshowEditorAdapter. При этом также обновляется ListView из SlideshowEditor, поскольку был изменен набор данных.

Листинг 12.21. Слушатель deleteButtonListener метода OnClickListener, обрабатывающий события кнопки deleteButton, находящейся рядом с выбранным изображением

```
144 // вызывается после выбора пользователем кнопки "Delete" Button,
145 // рядом с ImageView
146 private OnClickListener deleteButtonListener = new OnClickListener()
147 {
148    // удаляет изображение
149    @Override
150    public void onClick(View v)
```

```
      {
         slideshowEditorAdapter.remove((String) v.getTag());
      }  // конец описания метода onClick
   }; // конец описания слушателя OnClickListener deleteButtonListener

```

Частные классы ViewHolder и SlideshowEditorAdaptor: отображение изображений слайд-шоу с помощью шаблона View-Holder

Как было отмечено в листинге 12.7, при отображении элементов ListView в SlideshowEditor использовался шаблон view-holder. Класс ViewHolder (см. листинг 12.22, строки 158–162) определяет два компонента GUI, применяемые в каждом компоненте ListView. Класс SlideshowEditorAdapter (строки 165–212) расширяет класс ArrayAdapter для отображения каждого изображения из слайд-шоу в виде элемента ListView из SlideshowEditor. Элементы List, инициализируемые в конструкторе, включают строки Strings, представляющие локации изображений слайд-шоу. Код для SlideshowEditorAdapter подобен коду для SlideshowAdapter, приведенному в листинге 12.7, но данный адаптер использует разметку slideshow_edit_item.xml для элементов ListView. Дополнительные сведения о порядке отображения каждого изображения приведены в обсуждении листинга 12.7.

Листинг 12.22. Частный вложенный класс SlideshowEditorAdapter отображает изображения слайд-шоу с помощью компонента ListView из SlideshowEditor

```
156   // Класс, реализующий "шаблон ViewHolder pattern",
157   // обеспечивающий лучшую производительность ListView
158   private static class ViewHolder
159   {
160      ImageView slideImageView; // ссылка на ImageView элемента ListView
161      Button deleteButton; // ссылка на Button элемента ListView
162   }  // конец определения класса ViewHolder
163
164   // ArrayAdapter, отображающий изображения Slideshow
165   private class SlideshowEditorAdapter extends ArrayAdapter<String>
166   {
167      private List<String> items; // список изображений для Uri
168      private LayoutInflater inflater;
169
170      public SlideshowEditorAdapter(Context context, List<String> items)
171      {
172         super(context, -1, items);
173         this.items = items;
174         inflater = (LayoutInflater)
175         getSystemService(Context.LAYOUT_INFLATER_SERVICE);
176      }  // конец определения конструктора SlideshoweditorAdapter
177
178      @Override
179    public View getView(int position, View convertView, ViewGroup parent)
180      {
181          ViewHolder viewHolder; // ссылки на GUI текущего элемента
182
```

продолжение ⇨

Листинг 12.22 *(продолжение)*

```
      // если convertView равен null, «раздувание» и создание ViewHolder;
      // иначе получить существующий шаблон ViewHolder
      if (convertView == null)
      {
         convertView =
            inflater.inflate(R.layout.slideshow_edit_item, null);

         // настройка ViewHolder для этого элемента ListView
         viewHolder = new ViewHolder();
         viewHolder.slideImageView = (ImageView)
         convertView.findViewById(R.id.slideshowImageView);
         viewHolder.deleteButton =
            (Button) convertView.findViewById(R.id.deleteButton);
         convertView.setTag(viewHolder); // сохранение в виде тега View
      }  // конец блока if
      else // получение ViewHolder из тега convertView
         viewHolder = (ViewHolder) convertView.getTag();

      // получение и отображение изображения Bitmap миниатюры
      String item = items.get(position); // получение
                                         // текущего изображения
      new LoadThumbnailTask().execute(viewHolder.slideImageView,
         Uri.parse(item));

      // конфигурирование кнопки "Delete" Button
      viewHolder.deleteButton.setTag(item);
      viewHolder.deleteButton.setOnClickListener(deleteButtonListener);

      return convertView;
    } // конец определения метода getView
  } // конец определения класса SlideshowEditorAdapter
```

Вложенный класс LoadThumbnailTask

Класс LoadThumbnailTask (см. листинг 12.23) загружает миниатюру изображения в отдельном потоке выполнения, что позволяет гарантировать сохранение «отзывчивости» потока GUI. Метод doInBackground использует статический метод утилиты getThumbnail класса Slideshow для загрузки миниатюры. После завершения загрузки метод onPostExecute принимает растр (Bitmap) миниатюры и отображает его в указанном ImageView.

Листинг 12.23. Класс LoadThumbnailTask загружает миниатюру изображения в отдельный поток

```
   // задача по загрузке миниатюр в отдельном потоке
  private class LoadThumbnailTask extends AsyncTask<Object,Object,Bitmap>
   {
   ImageView imageView; // отображение миниатюры

```

```
    // загрузка миниатюры с аргументами ImageView, MediaType и Uri
    @Override
    protected Bitmap doInBackground(Object... params)
    {
       imageView = (ImageView) params[0];

       return Slideshow.getThumbnail((Uri) params[1],
          getContentResolver(), new BitmapFactory.Options());
    }  // конец определения метода doInBackground

    // установка миниатюры на ListView
    @Override
    protected void onPostExecute(Bitmap result)
       {
          super.onPostExecute(result);
          imageView.setImageBitmap(result);
       }  // конец определения метода onPostExecute
    }  // конец определения метода LoadThumbnailTask
 }  // конец определения класса SlideshowEditor
```

12.5.4. Подкласс SlideshowPlayer класса ListActivity

Подкласс SlideshowPlayer класса Activity (см. листинги 12.24–12.28) отображает слайд-шоу, заданное как дополнение к объекту Intent, который запускает этот класс Activity.

Операторы package, import и поля экземпляра класса SlideshowPlayer

В листинге 12.24 начинается определение класса SlideShowPlayer. Были выделены операторы import для новых классов и интерфейсов, рассмотренных в разделе 12.3 и в текущем разделе. С помощью константы String, указанной в строке 25, выполняется регистрация сообщений об ошибках, которые могут иметь место в случае воспроизведения фоновой музыки во время демонстрации слайд-шоу. Константы String, определенные в строках 28–30, применяются для сохранения информации о состоянии в onSaveInstanceState и для загрузки этой информации в метод onCreate в случаях, когда Activity переходит в режим фонового выполнения и возвращается на передний план соответственно. Константа int в строке 32 определяет длительность отображения каждого слайда в слайд-шоу. В строках 33–40 объявляются перенесенные экземпляры класса, использующиеся для управления слайд-шоу.

Листинг 12.24. Операторы package, import и поля экземпляра класса SlideshowPlayer

```
1  // SlideshowPlayer.java
2  // Воспроизведение выбранных слайд-шоу, передаваемых в виде
   // дополнений объекта Intent
3  package com.deitel.slideshow;
4
5  import java.io.FileNotFoundException;
6  import java.io.InputStream;
7
8  import android.app.Activity;
```

продолжение ⇨

Листинг 12.24 *(продолжение)*

```
import android.content.ContentResolver;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.TransitionDrawable;
import android.media.MediaPlayer;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.widget.ImageView;

public class SlideshowPlayer extends Activity
{
   private static final String TAG = "SLIDESHOW";  // тег регистрации
                                                    // ошибок

   // константы, используемые для сохранения состояния слайд-шоу при
   // изменении конфигурации
   private static final String MEDIA_TIME = "MEDIA_TIME";
   private static final String IMAGE_INDEX = "IMAGE_INDEX";
   private static final String SLIDESHOW_NAME = "SLIDESHOW_NAME";

   private static final int DURATION = 5000; // 5 секунд на слайд
   private ImageView imageView;  // отображает текущее изображение
   private String slideshowName; // имя текущего слайд-шоу
   private SlideshowInfo slideshow; // воспроизводимое слайд-шоу
   private BitmapFactory.Options options; // параметры загрузки
                                          // изображений
   private Handler handler; // используется для обновления слайд-шоу
   private int nextItemIndex; // индекс следующего
                              // отображаемого изображения
   private int mediaTime; // время (в миллисекундах)
                          // для медиаклипа
   private MediaPlayer mediaPlayer; // воспроизведение фоновой музыки

```

Переопределение метода onCreate класса Activity

В листинге 12.25 переопределяется метод onCreate класса Activity для конфигурирования SlideshowPlayer. В строке 49 вызывается ImageView из SlideshowPlayer. В строках 51–68 определяется, запустился ли Activity «с нуля» (в этом случае значение savedInstanceState Bundle равно null (строка 51)), либо Activity перезапускается (возможно по причине изменения конфигурации). Если Activity запускается «с нуля», в строке 54 получаем имя слайд-шоу из Intent, который запустил этот Activity. В строке 55 mediaTime присваивается 0, чтобы показать возможность играть музыку с самого начала. В строке 56 nextItemIndex присваивается значение 0, чтобы показать возможность запуска слайд-шоу «с

нуля». Если Activity перезапущен, в строках 61–67 значения присваиваются переменным из экземпляра класса значений, сохраненным в savedInstanceState Bundle.

Листинг 12.25. Переопределение метода onCreate класса Activity

```
42  // инициализация подкласса Activieн класса SlideshowPlayer
43  @Override
44  public void onCreate(Bundle savedInstanceState)
45  {
46      super.onCreate(savedInstanceState);
47      setContentView(R.layout.slideshow_player);
48
49      imageView = (ImageView) findViewById(R.id.imageView);
50
51      if (savedInstanceState == null)
52      {
53         // получение имени слайд-шоу из дополнения Intent
54        slideshowName = getIntent().getStringExtra(Slideshow.NAME_EXTRA);
55         mediaTime = 0;     // позиция в медиаклипе
56         nextItemIndex = 0; // начало с первого изображения
57      }  // конец блока if
58      else // восстановление Activity
59      {
60         // получение следующей позиции, сохраненной
           // при изменении конфигурации
61         mediaTime = savedInstanceState.getInt(MEDIA_TIME);
62
63         // получение индекса изображения, которое было отображено
           // при изменении конфигурации
64         nextItemIndex = savedInstanceState.getInt(IMAGE_INDEX);
65
66         // получение имени слайд-шоу, которое воспроизводилось
           // при изменении конфигурации
67         slideshowName = savedInstanceState.getString(SLIDESHOW_NAME);
68      }  // конец else
69
70      // получение SlideshowInfo для воспроизводимого слайд-шоу
71      slideshow = Slideshow.getSlideshowInfo(slideshowName);
72
73      // конфигурирование BitmapFactory.Options для загрузки изображений
74      options = new BitmapFactory.Options();
75      options.inSampleSize = 4; // образец, размеры которого равны
                                  // 1/4 размеров исходного изображения
76
77      // если есть фоновая музыка
78      if (slideshow.getMusicPath() != null)
79      {
80         // блок try для MediaPlayer, воспроизводящего музыку
81         try
82         {
83            mediaPlayer = new MediaPlayer();
```

продолжение ➩

Листинг 12.25 *(продолжение)*

```
84             mediaPlayer.setDataSource(
85             this, Uri.parse(slideshow.getMusicPath()));
86             mediaPlayer.prepare(); // подготовка MediaPlayer
                                      // для воспроизведения
87             mediaPlayer.setLooping(true); // циклическое воспроизведение
                                             // музыки
88             mediaPlayer.seekTo(mediaTime); // установка на mediaTime
89          }  // end try
90          catch (Exception e)
91          {
92             Log.v(TAG, e.toString());
93          }  // конец блока catch
94       }  // конец блока if
95
96       handler = new Handler(); // создание обработчика
                                  // для контроля слайд-шоу
97   }   // конец определения метода onCreate
98
```

В строке 71 получаем объект SlideshowInfo, относящийся к воспроизводимому слайд-шоу, а в строках 74–75 конфигурируется объект BitmapFactory.Options, применяемый для понижения разрешения изображений, отображаемых слайд-шоу.

Если при демонстрации слайд-шоу воспроизводится фоновая музыка, в строке 83 создается объект MediaPlayer, используемый для воспроизведения музыки. В строках 84–85 вызывается метод setDataSource класса MediaPlayer с аргументом Uri, определяющим местоположение воспроизводимой музыки. Метод prepare класса MediaPlayer (строка 86) подготавливается MediaPlayer для воспроизведения музыки. Этот метод блокирует текущий поток до тех пор, пока MediaPlayer не будет готов к воспроизведению. Причем данный метод может применяться только для музыки, сохраненной в памяти устройства. Если воспроизводится потоковый медиафайл, рекомендуется воспользоваться методом prepareAsync, который возвращает результат незамедлительно. Метод prepare блокирует текущий поток выполнения до тех пор, пока не будет выполнена буферизация медиапотока. Метод prepare генерирует исключение, если MediaPlayer не может быть подготовлен (например, если этот объект воспроизводит в данное время медиаклип). В случае генерирования исключения выполняется регистрация сообщения об ошибке (строка 92). Подробную диаграмму состояния для класса MediaPlayer можно найти на веб-сайте developer.android.com/reference/android/media/MediaPlayer.html.

В строке 87 вызывается метод setLooping класса MediaPlayer с аргументом true. В результате, если длительность музыкального фрагмента меньше, чем общая длительность слайд-шоу, музыкальное сопровождение воспроизводится циклически. В строке 88 вызывается метод seekTo класса MediaPlayer, настраивающий воспроизведение звука с определенного момента времени, заданного в миллисекундах. Этот аргумент равен 0 в случае, если Activity начинается с начала записи; в противном случае аргумент будет указывать на позицию последней остановки воспроизведения. И наконец, в строке 96 создается объект Handler, контролирующий выполнение слайд-шоу.

Переопределение методов onStart, onPause, onResume, onStop и onDestroy класса Activity

В листинге 12.26 переопределяются методы onStart, onPause, onResume, onStop и onDestroy класса Activity. Метод onStart (строки 100–105) немедленно отправляет объект Runnable методу updateSlideshow (см. листинг 12.28) для выполнения. Метод onPause (строки 108–115) приостанавливает звучание фоновой музыки путем вызова метода pause класса MediaPlayer. При этом предотвращается воспроизведение музыки в случае, если класс Activity не воспроизводится на переднем плане. Метод onResume (строки 118–125) вызывает метод start класса MediaPlayer, который начинает воспроизведение музыки либо возобновляет воспроизведение, если оно было приостановлено. Метод onStop (строки 128–135) вызывает метод обработчика removeCallbacks, который предотвращает выполнение предварительно запланированного Runnables из updateSlideshow в случае, если Activity приостановлен. Метод onDestroy (строки 138–145) вызывает метод release класса MediaPlayer, который освобождает ресурсы, используемые MediaPlayer.

Листинг 12.26. Переопределение методов onStart, onPause, onResume, onStop и onDestroy класса Activity

```
99   // вызывается после вызова метода onCreate и иногда метода onStop
100  @Override
101  protected void onStart()
102  {
103     super.onStart();
104     handler.post(updateSlideshow); // отсылает updateSlideshow
                                       // для выполнения
105  }  // конец определения метода onStart
106
107  // вызывается, если для Activity выбрана пауза
108  @Override
109  protected void onPause()
110  {
111     super.onPause();
112
113     if (mediaPlayer != null)
114     mediaPlayer.pause(); // воспроизведение приостановлено
115  }  // конец определения метода onPause
116
117  // вызывается после вызова метода onStart либо onPause
118  @Override
119  protected void onResume()
120  {
121     super.onResume();
122
123     if (mediaPlayer != null)
124     mediaPlayer.start(); // восстановление воспроизведения
125  }  // конец определения метода onResume
126
127  // вызывается, если Activity остановлен
128  @Override
```

продолжение ⇨

Листинг 12.26 *(продолжение)*

```
protected void onStop()
{
   super.onStop();

   // предотвращает воспроизведение слайд-шоу в фоновом режиме
   handler.removeCallbacks(updateSlideshow);
} // конец определения метода onStop

// вызывается при разрушении Activity
@Override
protected void onDestroy()
{
   super.onDestroy();

   if (mediaPlayer != null)
      mediaPlayer.release(); // освобождение ресурсов MediaPlayer
} // конец определения метода onDestroy

```

Переопределение метода onSaveInstanceState класса Activity

В листинге 2.27 переопределяется метод onSaveInstanceState таким образом, чтобы класс Activity сохранял позицию воспроизводимого слайд-шоу, индекс текущего изображения (минус единица, поскольку nextItemIndex фактически представляет следующее отображаемое изображение), а также имя слайд-шоу в Bundle из outState в случае изменения конфигурации устройства. Эта информация может быть восстановлена в методе onCreate для обеспечения продолжения воспроизведения слайд-шоу с момента изменения конфигурации устройства.

Листинг 12.27. Переопределение метода onSaveInstanceState класса Activity

```
// сохранение состояния слайд-шоу, которое может быть
// восстановлено в onCreate
@Override
protected void onSaveInstanceState(Bundle outState)
{
   super.onSaveInstanceState(outState);

   // если запущен mediaPlayer, сохранение текущей позиции медиафайла
   if (mediaPlayer != null)
      outState.putInt(MEDIA_TIME, mediaPlayer.getCurrentPosition());

   // сохранение nextItemIndex и slideshowName
   outState.putInt(IMAGE_INDEX, nextItemIndex – 1);
   outState.putString(SLIDESHOW_NAME, slideshowName);
} // конец определения метода onSaveInstanceState

```

Частный метод updateSlideshow класса Runnable

В листинге 12.28 определяется класс Runnable, который отображает изображения слайд-шоу. Если последнее изображение слайд-шоу уже отображено на экране (строка 168), в строках 171–172 происходит переустановка MediaPlayer, позволяющая освободить ресурсы. В строке 173 вызывается метод finish класса Activity, прерывающий выполнение текущего класса Activity и возвращающий к Activity, запущенной SlideshowPlayer.

Листинг 12.28. Метод updateSlideshow класса Runnable отображает следующее изображение в слайд-шоу и запускает самого себя снова через 5 секунд

```
162 // анонимный внутренний класс, реализующий объект Runnable
    // для управления слайд-шоу
163 private Runnable updateSlideshow = new Runnable()
164 {
165    @Override
166    public void run()
167    {
168       if (nextItemIndex >= slideshow.size())
169       {
170          // если воспроизводится музыка
171          if (mediaPlayer != null && mediaPlayer.isPlaying())
172             mediaPlayer.reset(); // слайд-шоу завершено,
                                     // сброс mediaPlayer
173             finish(); // возврат к запускающему Activity
174          }  // конец блока if
175          else
176          {
177             String item = slideshow.getImageAt(nextItemIndex);
178             new LoadImageTask().execute(Uri.parse(item));
179             ++nextItemIndex;
180          }  // конец блока else
181       }  // конец определения метода run
182
183       // задача по загрузке миниатюр в отдельный поток
184       class LoadImageTask extends AsyncTask<Uri, Object, Bitmap>
185       {
186          // загрузка изображений
187          @Override
188          protected Bitmap doInBackground(Uri... params)
189          {
190            return getBitmap(params[0], getContentResolver(), options);
191          }  // конец определения метода doInBackground
192
193          // настройка миниатюр в ListView
194          @Override
195          protected void onPostExecute(Bitmap result)
196          {
197             super.onPostExecute(result);
198             BitmapDrawable next = new BitmapDrawable(result);
199             next.setGravity(android.view.Gravity.CENTER);
```

продолжение ⇨

Листинг 12.28 *(продолжение)*

```
200             Drawable previous = imageView.getDrawable();
201
202             // если предыдущий объект TransitionDrawable,
203             // получение его второго элемента Drawable
204             if (previous instanceof TransitionDrawable)
205             previous = ((TransitionDrawable) previous).getDrawable(1);
206
207             if (previous == null)
208                imageView.setImageDrawable(next);
209             else
210             {
211                Drawable[] drawables = { previous, next };
212                TransitionDrawable transition =
213                   new TransitionDrawable(drawables);
214                imageView.setImageDrawable(transition);
215                transition.startTransition(1000);
216             }  // конец блока else
217
218             handler.postDelayed(updateSlideshow, DURATION);
219          }  // конец определения метода onPostExecute
220       }  // конец определения класса LoadImageTask
221
222       // метод утилиты, применяемый для получения Bitmap из Uri
223       public Bitmap getBitmap(Uri uri, ContentResolver cr,
224          BitmapFactory.Options options)
225       {
226          Bitmap bitmap = null;
227
228          // получение изображения
229          try
230          {
231             InputStream input = cr.openInputStream(uri);
232             bitmap = BitmapFactory.decodeStream(input, null, options);
233          }  // конец блока try
234          catch (FileNotFoundException e)
235          {
236             Log.v(TAG, e.toString());
237          }  // конец блока catch
238
239          return bitmap;
240       }  // конец определения метода getBitmap
241    }; // конец определения метода updateSlideshow из Runnable
242 }  // конец определения класса SlieshowPlayer
```

Если отображается несколько изображений, в строке 177 получаем путь к следующему изображению, а в строке 178 запускается задача LoadImageTask, выполняющая загрузку и отображение изображения. Класс LoadImageTask (строки 184–220) загружает следующее изображение и переходы от предыдущего изображения к следующему. Первый метод doInBackground вызывает метод getBitmap (определен в строках 223–240) для получения изображения. Если изображение возвращено, метод onPostExecute обрабатывает

переходы изображения. В строках 198–199 создается объект BitmapDrawable на основе возвращенного объекта Bitmap (результат), а его параметру gravity присваивается значение center. В результате изображение отображается в центре ImageView. В строке 200 получается ссылка на предыдущий объект Drawable. Если это TransitionDrawable, мы получаем второй объект BitmapDrawable класса TransitionDrawable. (Обратите внимание, что цепочка объектов TransitionDrawable, которая может привести к переполнению памяти, не создается.) Если предыдущий объект Drawable отсутствует, в строке 208 просто отображается новый объект BitmapDrawable. Если же такие объекты имеются, в строках 211–215 с помощью класса TransitionDrawable выполняется переход между двумя объектами Drawable в ImageView. В строке 214 объект TransitionDrawable передается методу setImageDrawable класса ImageView для его отображения на currentImageView. Мы создаем объект TransitionDrawable программным способом, как только возникает необходимость в динамическом определении предыдущего и следующего изображений. Метод startTransition класса TransitionDrawable (строка 215) выполняет переход примерно за одну секунду (1000 миллисекунд). Переход представляет собой наплыв, который выполняется автоматически между первым и вторым объектами Drawable в массиве drawables. В строке 218 запланирован вызов метода updateSlideshow, который вызывается каждые 5 секунд для отображения следующего изображения.

Функция getBitmap (строки 223–240) использует метод ContentResolver для получения потока ввода (InputSteam) для указанного изображения. Затем в строке 232 применяется статический метод decodeStream класса BitmapFactory для создания объекта Bitmap из данного потока. В качестве аргументов этого метода используются InputStream, предназначенный для чтения изображения, Rect — для формирования отступов вокруг изображения (значение null означает отсутствие отступов) и объект BitmapFactory.Options, определяющий способ понижения разрешения изображения.

12.6. Резюме

В этой главе вы создавали приложение Slideshow, с помощью которого пользователи могут создавать слайд-шоу и управлять ими. Вы изучили порядок использования провайдеров контента Android, применяемых для сохранения и выборки данных в приложения, а также для обеспечения доступа к данным для различных приложений. Также использовались встроенные провайдеры, с помощью которых пользователи могут выбирать изображения и аудиофайлы, хранящиеся в памяти устройства. Чтобы воспользоваться преимуществами провайдеров контента, запускались объекты Intent и определялся тип MIME для требуемых данных. Затем Android запускал Activity, который отображал пользователю выбранный тип данных либо выводил диалоговое окно выбора Activity, в котором пользователь выбирал используемый объект Activity.

Для получения данных, вводимых пользователем, использовались диалоговые окна AlertDialog с выбранными пользовательскими представлениями View. Также была настроена разметка ListActivity путем замены заданной по умолчанию разметки на одну из разметок, находящихся в ListView, атрибуту android:id которого присвоено значение "@android:id/list". Также с помощью подклассов класса ArrayAdapter были созданы объекты, заполняющие ListViews данными объектов коллекций. Как только изменялись наборы данных ArrayAdapter, вызывался метод notifyDataSetChanged, обновляющий

соответствующий объект ListView. Вы научились использовать шаблон view-holder для увеличения быстродействия ListViews со сложными структурами элементов списка.

В ходе создания приложения использовать объекты Intent для запуска Activity, который возвращает результат, а также обрабатывает результат в случае возврата Activity. С помощью метода setTag класса View выполнялось добавление объекта Object в представление View, причем этот объект использовался позднее обработчиками событий.

Вы использовали объект MediaPlayer для воспроизведения музыки из аудофайлов, которые хранятся на устройстве. С помощью объекта BitmapFactory создавались объекты Bitmap на основе настроек, определяемых объектом BitmapFactory.Options. И наконец, добавлялись переходы между изображениями с помощью объектов TransitionDrawable, отображаемых на ImageView.

В главе 13 будет создано приложение Enhanced Slideshow, при выполнении которого пользователь может создавать фотографии с помощью камеры, добавлять в слайд-шоу видеоролики и сохранять слайд-шоу в памяти устройства.

13 Приложение Enhanced Slideshow App

Сериализация данных, фотографирование с помощью приложения Camera и воспроизведение видеороликов с помощью VideoView

В этой главе...

- Использование объектов Intent и распознавателей контента для выбора видеороликов из медиатеки устройства.
- Создание новых фотографий с помощью тыльной камеры устройства, добавляемых в слайд-шоу.
- Использование объектов SurfaceView, SurfaceHolder и Camera для предварительного просмотра фотографий с применением различных цветовых эффектов.
- Воспроизведение видеороликов с помощью компонентов VideoView.
- Сохранение и загрузка слайд-шоу с помощью объектов Serializable.
- Сохранение слайд-шоу с помощью потоков ObjectOutputStream и FileOutputStream.
- Загрузка слайд-шоу с устройства с помощью потоков ObjectInputStream и FileInputStream.

13.1. Введение

Приложение Enhanced Slideshow включает несколько новых функций по сравнению с описанным в главе 12 приложением Slideshow. Благодаря новой версии приложения

пользователь может *сохранять* содержимое слайд-шоу в памяти устройства, используя *обработку файлов* и *сериализацию объекта*. В результате слайд-шоу становятся доступными для воспроизведения после запуска приложения. К тому же при редактировании слайд-шоу пользователь может создавать новые фотографии с помощью камеры устройства (по умолчанию выбирается тыльная камера; см. рис. 13.1). Также можно выбрать видеоролики, хранящиеся в памяти устройства, и включить их в слайд-шоу (см. рис. 13.2, *а*). Как и в случае с изображениями, после выбора видеоролика пользователем миниатюра (см. рис. 13.2, *б*) отображается в списке элементов, включенных в слайд-шоу. Как только объект SlideshowPlayer Activity распознает видеоролик (см. рис. 13.2), он начинает воспроизводить его с помощью VideoView во время воспроизведения музыки слайд-шоу в фоновом режиме.

ПРИМЕЧАНИЕ

Чтобы протестировать возможности фото- и видеосъемки приложения, потребуется физическое устройство Android. На момент написания этой книги эмулятор Android не поддерживал функции камеры, а возможности эмуляции воспроизведения видео были весьма ограничены.

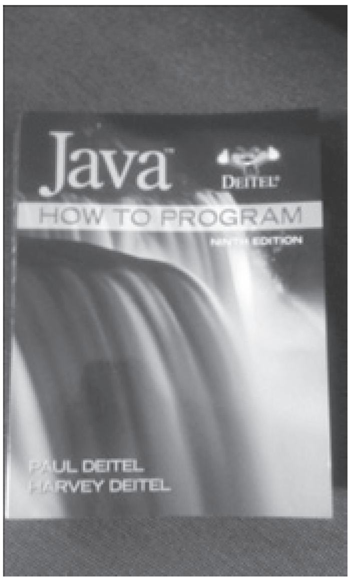
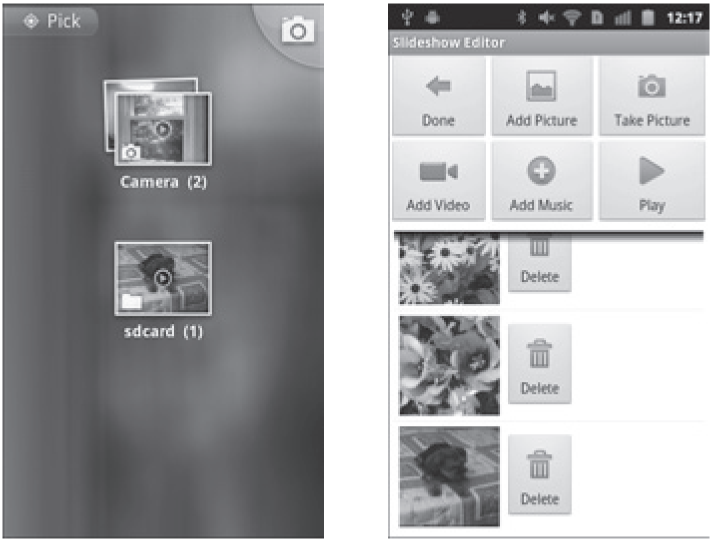


Рис. 13.1. Предварительный просмотр новой фотографии, сделанной с помощью камеры

13.2. Тестирование приложения Enhanced Slideshow App

Открытие и выполнение приложения

Откройте среду Eclipse и импортируйте проект Enhanced Slideshow app. Выполните следующие действия:



а *б*

Рис. 13.2. Выбор видеоролика и отображение миниатюры после выделения видео: *а* — выбор видеоролика в памяти устройства; *б* — миниатюра видео после выделения

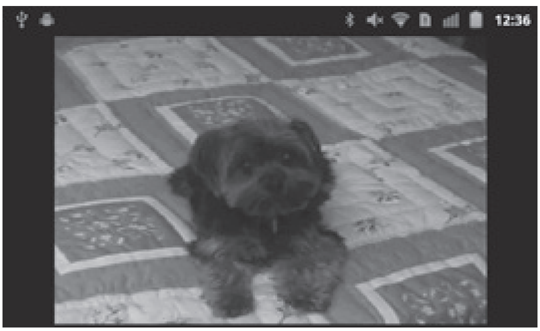


Рис. 13.3. Воспроизведение видеоролика в VideoView после выбора для устройства альбомного режима

1. Выполните команды File ▸ Import... (Файл ▸ Импорт...) для отображения диалогового окна Import (Импорт).
2. Раскройте узел General (Общий) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область). Затем щелкните на кнопке Next > (Далее >).
3. Справа от текстового поля выберите параметр Select root directory (Выбрать корневой каталог) и щелкните на кнопке Browse... (Просмотр...). В диалоговом окне Browse For Folder (Просмотр папки) выберите папку EnhancedSlideshow в папке примеров книги и щелкните на кнопке OK.
4. Щелкните на кнопке Finish (Готово) для импорта проекта в среду Eclipse.

В среде Eclipse щелкните правой кнопкой мыши на проекте приложения, находящемся в окне Package Explorer, затем в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android).

Добавление видео на устройство AVD

Выполните описанные в разделе 12.2 действия для добавления изображений и аудиофайлов в память устройства AVD, чтобы добавить образцы видеороликов в папку примеров книги.

ПРИМЕЧАНИЕ

Как отмечалось ранее, эмулятор не поддерживает воспроизведение видеофайлов, поэтому это приложение следует тестировать на реальном устройстве.

Добавление и изменение слайд-шоу

Как и в главе 12, коснитесь кнопки меню устройства, затем выберите пункт меню New Slideshow (Создать слайд-шоу) для отображения диалогового окна Set Slideshow Name (Присвоить имя слайд-шоу). Выберите имя для слайд-шоу, затем коснитесь кнопки Set Name для создания слайд-шоу и отображения окна Slideshow Editor.

Измените слайд-шоу (см. главу 12). В текущей версии приложения протестируйте функции добавления видеороликов и создания фотоснимков. После завершения изменения слайд-шоу и выбора кнопки Done приложение вернется к основному классу Slideshow Activity, выполняющему сохранение слайд-шоу в памяти устройства.

ПРИМЕЧАНИЕ

В этом приложении выполняется сохранение слайд-шоу после возвращения к главному экрану приложения после изменения слайд-шоу. Также можно сконфигурировать приложение после завершения изменения слайд-шоу в Slideshow Editor.

Воспроизведение слайд-шоу

Когда в процессе воспроизведения деятельность SlideshowPlayer встречает видео, она воспроизводит его с помощью компонента VideoView, в то время как музыкальное сопровождение слайд-шоу продолжает воспроизводиться в фоновом режиме.

13.3. Обзор применяемых технологий

В этом разделе представлены новые технологии, применяемые при создании приложения Enhanced Slideshow.

Обработка файла и сериализация объекта

В приложении сохраняются слайд-шоу, которые будут просмотрены позже. В создаваемых ранее приложениях были использованы методики, предусматривающие сохранение текстовых данных в парах «ключ–значение». В этом приложении сохраняются объекты SlideshowInfo с помощью *сериализации объекта* (раздел 13.5.3). Сериализованный

объект представлен в виде последовательности байтов, включающей данные объекта и информацию о типе объекта.

Функции сериализации определяются с помощью пакета java.io. Чтобы использовать объект, в который встроены функции сериализации, класс объекта должен реализовать интерфейс Serializable, представляющий собой тегирующий интерфейс. Подобные интерфейсы не включают методы. Объекты класса, которые реализуют интерфейс Serializable, тегируются как объекты Serializable. Таким образом, объект класса, реализующий интерфейс Serializable, является объектом Serializable. Класс ObjectOutputStream сериализирует объекты Serializable в заданный поток OutputSteam — в рассматриваемом приложении это поток FileOutputStream. Тегирование объектов как Serializable обязательно, поскольку вывод потока ObjectOutputStream содержит только объекты Serializable.

Это приложение сериализирует все объекты List of SlideshowInfo путем передачи объекта List методу writeObject класса ObjectOutputStream. Этот метод создает объект graph, который включает объект List. Этот объект включает ссылки на все объекты SlideshowInfo, а также на объекты, на который ссылаются объекты SlideshowInfo, и на прочие подобные объекты. Если в объекте graph находится объект, не относящийся к типу Serializable, генерируется исключение NotSerializableException. В результате обеспечивается проверка описаний класса для классов библиотеки в документации, доступной в Интернете. В процессе подобной проверки определяется, будут ли сериализуемые объекты реализовывать интерфейс Serializable непосредственно либо наследовать соответствующую связь из суперкласса в иерархии.

Сериализуемый объект может считываться из файла и десериализироваться. В результате информация о типе и байты, представляющие объект и относящиеся к нему данные, используются для повторного создания объекта graph в памяти. Эта задача выполняется с помощью объекта ObjectInputStream, который считывает байты из указанного потока InputStream, — в этом приложении из потока FileInputStream. Метод readObject класса ObjectOutputStream возвращает десериализованный объект, имеющий тип Object. Чтобы использовать этот метод в приложении, следует привести объект к соответствующему типу — в данном приложении к типу List<SlideshowInfo>.

Использование тыльной камеры устройства для создания фотоснимков, хранящихся в приложении Gallery устройства

С помощью приложения Enhanced Slideshow пользователь может создать новый фотоснимок с помощью тыльной камеры устройства, сохранить его с помощью приложения Gallery устройства и добавить новую фотографию в слайд-шоу. В разделе 13.5.5 с помощью классов Camera (пакет android.hardware) и SurfaceView (пакет android.view) обеспечивается предварительный просмотр создаваемых фотографий. Как только пользователь коснется экрана, объект PictureTaker Activity передает инструкцию объекту Camera, определяющую создание фотоснимка. Затем передается уведомление объекту Camera.PictureCallback о том, что фотография была снята. Выполняется захват данных изображения, которые сохраняются в Gallery, а потом возвращаются соответствующие Uri в SlideshowEditor Activity, в результате чего новое изображение добавляется в слайд-шоу. Объект PictureTaker Activity также поддерживает меню, в котором пользователь может выбрать список поддерживаемых эффектов цвета объекта Camera. По умолчанию создаются цветные фотографии, хотя современные цифровые фотоаппараты могут

поддерживать различные цветовые эффекты, например *черно-белый*, *сепия* или *негатив*. Список поддерживаемых эффектов можно получить из объекта Camera.Parameters, связанного с объектом Camera. Обратите внимание, что можно запустить встроенный в камеру объект Activity, с помощью которого пользователь может создавать снимки, но в этой главе будет демонстрироваться непосредственный доступ к функциям камеры. Чтобы воспользоваться встроенным в камеру Activity, придерживайтесь следующих инструкций:

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
intput.putExtra(MediaStore.EXTRA_OUTPUT, storageURI);
startActivityForResult(intent, requestCode);
```

Инструкция storageURI определяет местоположение сохраняемых фотографий. Вы можете переопределить onActivityResult, чтобы проверить requestCode и обработать результаты, возвращаемые деятельностью встроенной камеры.

Выбор видеороликов, воспроизводимых в слайд-шоу

Приложение Slideshow применялось в качестве объекта Intent для запуска Activity, с помощью которого выбирается изображение из Gallery. Аналогичная технология применяется в этом приложении (раздел 13.5.4) для выбора пользователем видеороликов. В данном случае определяется другой тип MIME для данных, в результате чего отображаются только видеофильмы.

Воспроизведение видеороликов с помощью VideoView

Класс SlideshowPlayer Activity из приложения Enhanced Slideshow (раздел 13.5.6) использует компонент VideoView для воспроизведения видеороликов из слайд-шоу. Путем определения URI видеороликов из VideoView указывается местоположение воспроизводимого видеоролика, а также объект MediaController (пакет android.widget), с помощью которого обеспечивается управление воспроизводимым видеороликом. Компонент VideoView поддерживает собственный объект MediaPlayer, используемый для воспроизведения видеороликов. С помощью слушателя MediaPlayer.OnCompletionListener определяется момент завершения воспроизведения видеоролика, после чего можно продолжить воспроизведение слайд-шоу, отображая следующее изображение или видеоролик.

13.4. Создание графического интерфейса пользователя и файлов ресурсов

В этом разделе будут рассмотрены изменения, внесенные в ресурсы и разметку GUI приложения Slideshow (глава 12). В результате будет создано приложение Enhanced Slideshow. Для просмотра содержимого файлов ресурсов откройте их в Eclipse.

13.4.1. Создание проекта

Вместо создания приложения «с нуля» скопируйте приложение Slideshow (глава 12) и переименуйте его следующим образом:

1. Скопируйте папку Slideshow и переименуйте ее в EnhancedSlideshow.

2. Импортируйте проект из папки EnhancedSlideshow в среду Eclipse.
3. Раскройте узел проекта src.
4. Правой кнопкой мыши щелкните на пакете com.deitel.slideshow и выполните команды Refactor ▸ Rename... (Рефакторинг ▸ Переименовать...).
5. В диалоговом окне Rename Package (Переименовать пакет) введите com.deitel.enhancedslideshow, а потом щелкните на кнопке Preview> (Просмотр>).
6. Щелкните на кнопке OK, чтобы изменить имя пакета во всем проекте.
7. В файле ресурсов strings.xml измените значение строкового ресурса app_name на "Enhanced Slideshow".

13.4.2. Файл AndroidManifest.xml

В листинге 13.1 приводится код файла ресурсов AndroidManifest.xml. Был добавлен элемент activity в новом классе PictureTaker Activity (строки 26–29), а также указано на то, что приложение требует разрешения WRITE_EXTERNAL_STORAGE и CAMERA.

Листинг13.1. AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android=http://schemas.android.com/apk/res/android
3     package="com.deitel.enhancedslideshow" android:versionCode="1"
4     android:versionName="1.0">
5     <application android:icon="@drawable/icon"
6        android:label="@string/app_name"
7        android:debuggable="true">
8        <activity android:name=".Slideshow"
9           android:label="@string/app_name"
10          android:screenOrientation="portrait"
11          android:theme="@android:style/Theme.Light">
12          <intent-filter>
13             <action android:name="android.intent.action.MAIN" />
14             <category android:name="android.intent.category.LAUNCHER" />
15          </intent-filter>
16       </activity>
17
18       <activity android:name=".SlideshowEditor"
19          android:label="@string/slideshow_editor"
20          android:screenOrientation="portrait"></activity>
21
22       <activity android:name=".SlideshowPlayer"
23          android:label="@string/app_name"
24          android:theme="@android:style/Theme.NoTitleBar"></activity>
25
26       <activity android:name=".PictureTaker"
27          android:label="@string/app_name"
28          android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
29          android:screenOrientation="landscape"></activity>
30       </application>
31       <uses-sdk android:minSdkVersion="8" />
```

продолжение ☞

Листинг 13.1 *(продолжение)*

```
      <uses-permission
         android:name="android.permission.WRITE_EXTERNAL_STORAGE">
      </uses-permission>
      <uses-permission android:name="android.permission.CAMERA">
      </uses-permission>
</manifest>
```

13.4.3. Измененная разметка SlideshowEditor из ListActivity

На рис. 13.4 показана измененная разметка компонента ListActivity из SlideshowEditor, включающая два ряда кнопок Buttons в TableLayout, находящихся в верхней части графического интерфейса пользователя.

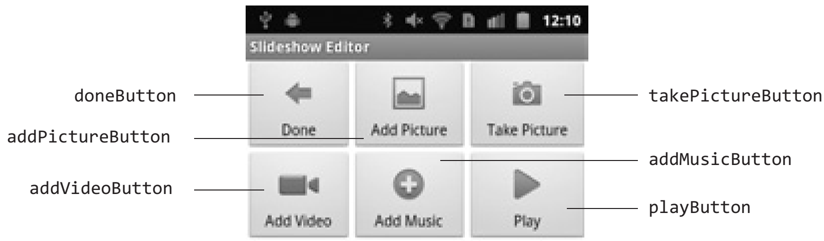


Рис. 13.4. Измененная разметка для ListActivity из SlideshowEditor, определенная в файле slideshow_editor.xml

13.4.4. Разметка PictureTaker класса Activity

В листинге 13.2 приводится XML-разметка объекта PictureTaker класса Activity (camera_preview.xml), включающего компонент SurfaceView, заполняющий экран. Компонент SurfaceView отображает изображение предварительного просмотра камеры после того, как пользователь завершил подготовку к фотосъемке.

Листинг 13.2. Разметка объекта PictureTaker класса Activity, определенная в файле camera_preview.xml

```
xml version="1.0" encoding="utf-8"?>
urfaceView xmlns:android=http://schemas.android.com/apk/res/android
   android:id="@+id/cameraSurfaceView"
      android:layout_width="match_parent"
   android:layout_height="match_parent">
</SurfaceView>
```

13.4.5. Измененная разметка SlideshowPlayer класса Activity

В листинге 13.3 показан код файла разметки slideshow_player.xml, представляющий собой измененную XML-разметку объекта SlideshowPlayer класса Activity. В приложении, разрабатываемом в этой главе, отображается компонент ImageView либо VideoView (в зависимости от выбранного текущего элемента слайд-шоу — изображение или видеоролик).

Поэтому выбирается макет FrameLayout, включающий компоненты ImageView и VideoView, которые занимают весь экран. В соответствии с отображаемыми в данный момент объектами отображаются либо скрываются соответствующие компоненты View.

Листинг 13.3. Измененная разметка для объекта SlideshowPlayer класса Activity, определенная в файле slideshow_editor.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android=http://schemas.android.com/apk/res/android
   android:layout_width="match_parent"
   android:layout_height="match_parent">
 <ImageView android:id="@+id/imageView" android:scaleType="centerInside"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_gravity="center"></ImageView>
  <VideoView android:id="@+id/videoView" android:layout_gravity="center"
      android:layout_width="match_parent"
      android:layout_height="match_parent"></VideoView>
 </FrameLayout>
```

13.5. Создание приложения

Это приложение включает классы MediaItem (листинг 13.4), SlideshowInfo (листинг 13.5), Slideshow (листинги 13.6–13.11), SlideshowEditor (листинги 13.12–13.14), PictureTaker (листинги 13.15–13.20) и SlideshowPlayer (листинги 13.21–13.23). Для классов из главы 12, которые были изменены, будут показаны лишь изменения.

13.5.1. Класс MediaItem

В приложении Slideshow местоположение каждого изображения сохраняется в объекте List<String>, который поддерживается компонентом ListActivity класса Slideshow. Чтобы обеспечить возможность включения изображений и видеороликов в приложение, был создан класс MediaItem (см. листинг 13.1), сохраняющий объекты MediaType и String. Объект enum MediaType (строка 12) включает константы, с помощью которых определяются объекты, представляемые MediaItem: изображение или видеоролик. Класс SlideshowInfo (раздел 13.5.2) поддерживает список List of MediaItems, представляющий все изображения и видеоролики, включенные в состав слайд-шоу. Поскольку приложение Enhanced Slideshow сериализует объекты SlideshowInfo, чтобы воспроизводить их в будущем, класс MediaItem реализует интерфейс Serializable.

Листинг 13.4. Класс MediaItem, применяемый для представления изображений и видеороликов в слайд-шоу

```
1  // MediaItem.java
2  // Представляет изображение или видеоролик в слайд-шоу.
3  package com.deitel.enhancedslideshow;
4
5  import java.io.Serializable;
```

продолжение ⇗

Листинг 13.4 *(продолжение)*

```

public class MediaItem implements Serializable
{
   private static final long serialVersionUID = 1L; // версия # класса

   // константы, определяющие типы медиаресурсов
   public static enum MediaType { IMAGE, VIDEO }

   private final MediaType type;  // в качестве этого MediaItem
                                  // выбирается IMAGE или VIDEO
   private final String path;     // местоположение MediaItem

   // конструктор
   public MediaItem(MediaType mediaType, String location)
   {
      type = mediaType;
      path = location;
   }  // конец определения конструктора

   // получение типа MediaType для изображения или видеоролика
   public MediaType getType()
   {
      return type;
   }  // конец определения метода MediaType

   // возвращение описания изображения или видеоролика
   public String getPath()
   {
      return path;
   }  // конец описания метода getDescription
}  // конец определения класса MediaItem
```

13.5.2. Класс SlideshowInfo

Класс SlideshowInfo этого приложения (см. листинг 13.5) был изменен для организации хранения списка List<MediaItem> (строка 13), представляющего местоположения изображения и видеоролика, а также тип каждого элемента (в то время как объект List<String> представляет лишь местоположения изображения). Методы getImageList, addImage и getImageAt были переименованы и получили имена getMediaItemList (строка 31), addMediaItem (строка 37) и getMediaItemAt (строка 43) соответственно. Каждый полученный метод манипулирует объектами MediaItems, а не Strings. Для поддержки сериализации класс SlideshowInfo реализует интерфейс Serializable (строка 9).

Листинг 13.5. Измененный класс SlideshowInfo хранит список List of MediaItems

```
1  // SlideshowInfo.java
2  // Хранятся данные для одного слайд-шоу.
3  package com.deitel.enhancedslideshow;
4
```

```java
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class SlideshowInfo implements Serializable
{
   private static final long serialVersionUID = 1L; // # версии класса
   private String name; // имя слайд-шоу
   private List<MediaItem> mediaItemList; // изображения слайд-шоу
   private String musicPath; // местоположение воспроизводимой музыки

   // конструктор
   public SlideshowInfo(String slideshowName)
   {
      name = slideshowName; // присваивание имени слайд-шоу
      mediaItemList = new ArrayList<MediaItem>();
      musicPath = null; // отсутствует музыка для слайд-шоу
   }  // конец описания конструктора SlideshowInfo

   // возвращает имя слайд-шоу
   public String getName()
   {
      return name;
   }  // конец описания метода getName

   // список List of MediaItems, указывающий на изображения слайд-шоу
   public List<MediaItem> getMediaItemList()
   {
      return mediaItemList;
   }  // конец определения метода getMediaItemList

   // добавление нового элемента MediaItem
   public void addMediaItem(MediaItem.MediaType type, String path)
   {
      mediaItemList.add(new MediaItem(type, path));
   }  // конец описания метода addMediaItem

   // возвращает MediaItem в позиции индекса
   public MediaItem getMediaItemAt(int index)
   {
      if (index >= 0 && index < mediaItemList.size())
         return mediaItemList.get(index);
      else
         return null;
   }  // конец описания метода getMediaItemAt

   // возврат музыки слайд-шоу
   public String getMusicPath()
   {
      return musicPath;
   }  // конец описания метода getMusicPath
```

продолжение ⇨

Листинг 13.5 *(продолжение)*

```

   // настройка музыки слайд-шоу
   public void setMusicPath(String path)
   {
      musicPath = path;
   }  // конец описания метода setMusicPath

   // возвращает количество изображений/видеороликов в слайд-шоу
   public int size()
   {
      return mediaItemList.size();
   }  // конец описания метода size
}  // конец описания класса SlideshowInfo
```

13.5.3. Класс Slideshow

При выполнении этого приложения выполняется сохранение слайд-шоу на устройстве для воспроизведения в будущем. Как отмечалось в разделе 13.3, для сохранения информации слайд-шоу используется сериализация объекта. Класс Slideshow (см. листинги 13.6–13.11), главный класс Activity приложения, был изменен таким образом, что получил возможность сохранения и загрузки объекта List<SlideshowInfo>. В этом разделе будут лишь представлены изменения класса Slideshow.

Операторы package, import и поля экземпляра класса Slideshow

Подкласс Slideshow класса ListActivity (см. листинг 13.6) получил несколько новых операторов import и новую переменную экземпляра класса. Новые свойства выделены в коде другим цветом. В строках 5–9 и 24 находятся классы import, используемые для обработки файлов и выполнения сериализации в приложении. Переменная экземпляра класса slideshowFile (строка 53) представляет местоположение файла приложения на устройстве.

Листинг 13.6. Операторы package, import и переменные экземпляра класса Slideshow

```
 1 // Slideshow.java
 2 // Главный класс Activity для класса Slideshow.
 3 package com.deitel.enhancedslideshow;
 4
 5 import java.io.File;
 6 import java.io.FileInputStream;
 7 import java.io.FileOutputStream;
 8 import java.io.ObjectInputStream;
 9 import java.io.ObjectOutputStream;
10 import java.util.ArrayList;
11 import java.util.List;
12
13 import android.app.AlertDialog;
14 import android.app.ListActivity;
15 import android.content.ContentResolver;
```

```
16 import android.content.Context;
17 import android.content.DialogInterface;
18 import android.content.Intent;
19 import android.graphics.Bitmap;
20 import android.graphics.BitmapFactory;
21 import android.net.Uri;
22 import android.os.AsyncTask;
23 import android.os.Bundle;
24 import android.provider.MediaStore;
25 import android.util.Log;
26 import android.view.Gravity;
27 import android.view.LayoutInflater;
28 import android.view.Menu;
29 import android.view.MenuInflater;
30 import android.view.MenuItem;
31 import android.view.View;
32 import android.view.View.OnClickListener;
33 import android.view.ViewGroup;
34 import android.widget.ArrayAdapter;
35 import android.widget.Button;
36 import android.widget.EditText;
37 import android.widget.ImageView;
38 import android.widget.ListView;
39 import android.widget.TextView;
40 import android.widget.Toast;
41
42
43 public class Slideshow extends ListActivity
44 {
45   private static final String TAG = "SLIDESHOW";//тег ошибки регистрации
46
47    // используется при добавлении имени слайд-шоу в качестве
      // дополнения к Intent
48    public static final String NAME_EXTRA = "NAME";
49
50    static List<SlideshowInfo> slideshowList; // список слайд-шоу
51    private ListView slideshowListView;       // компонент ListView
                                                // класса ListActivity
52    private SlideshowAdapter slideshowAdapter; // адаптер для ListView
53    private File slideshowFile; // файл, представляющий местоположение
                                  // слайд-шоу
54
```

Переопределение метода onCreate класса Activity

Метод onCreate класса Slideshow (см. листинг 13.7) создает объект File (строки 63–65), представляющий место в файловой системе Android, в котором приложение хранит слайд-шоу. Класс Context поддерживает методы, используемые для доступа к файловой системе. Метод getExternalFilesDir этого класса возвращает объект File, представляющий каталог внешнего хранилища данных устройства, специфичный для приложения. Обычно в качестве подобного хранилища используется карта памяти SD, но в случае

отсутствия поддержки подобных карт устройством используется внутренняя память самого устройства. Файлы, созданные в этом местоположении, автоматически управляются системой, — при удалении приложения также удаляются относящиеся к нему файлы. Для объекта File вызывается метод getAbsolutePath, затем добавляется фрагмент /EnhancedSlideshowData.ser для создания пути к файлу, в котором приложение хранит слайд-шоу. (Учтите, что каталог, находящийся во внешней памяти устройства, может быть недоступен для приложения в силу ряда причин, например пользователь может вытащить из устройства карту SD.) В строке 66 создается объект подкласса AsyncTask класса LoadSlideshowsTask (см. листинг 13.8), а также вызывается относящийся к нему метод execute, выполняющий загрузку ранее сохраненных слайд-шоу (в случае наличия подобных слайд-шоу). При выполнении этой задачи не требуются какие-либо аргументы, поэтому методу execute передается значение null.

Листинг 13.7. Переопределение метода onCreate из Activity в классе Slideshow

```
55  // вызывается при первом создании activity
56  @Override
57  public void onCreate(Bundle savedInstanceState)
58  {
59     super.onCreate(savedInstanceState);
60     slideshowListView = getListView(); // получение встроенного ListView
61
62     // получение местоположения файла и запуск задачи
       // по загрузке слайд-шоу
63     slideshowFile = new File(
64     getExternalFilesDir(null).getAbsolutePath() +
65        "/EnhancedSlideshowData.ser");
66     new LoadSlideshowsTask().execute((Object[]) null);
67
68     // создание нового AlertDialog Builder
69     AlertDialog.Builder builder = new AlertDialog.Builder(this);
70     builder.setTitle(R.string.welcome_message_title);
71     builder.setMessage(R.string.welcome_message);
72     builder.setPositiveButton(R.string.button_ok, null);
73     builder.show();
74  }  // конец определения метода onCreate
75
```

Подкласс LoadSlideshowsTask класса AsyncTask

Метод doInBackground класса LoadSlideshowsTask (см. листинг 13.8) проверяет, существует ли файл EnhancedSlideshowData.ser (строка 84). Если этот файл существует, создается объект ObjectInputStream (строки 88–89). В строке 90 вызывается метод readObject класса ObjectInputStream, используемый для чтения объекта List<SlideshowInfo> класса slideshowFile. Если файл не существует либо при чтении файла генерируется исключение, в строке 115 создается новый объект List<SlideshowInfo>. В случае генерирования исключения в строках 94–110 используется метод runOnUiThread класса Activity для отображения сообщения Toast из потока UI, описывающего проблему. После завершения

выполнения фоновой задачи вызывается метод onPostExecute (строки 121–130) из потока UI, применяемый для настройки адаптера ListView класса Slideshow.

Листинг 13.8. Класс LoadSlideshowsTask десериализирует объект List<SlideshowInfo> из файла либо создает объект, если файл не существует

```
76  // Класс, применяемый для загрузки объекта List<SlideshowInfo>
    // из устройства
77 private class LoadSlideshowsTask extends AsyncTask<Object,Object,Object>
78  {
79     // загрузка из потока, который не является потоком GUI
80     @Override
81     protected Object doInBackground(Object... arg0)
82     {
83        // если файл существует - чтение из файла, иначе - создание файла
84        if (slideshowFile.exists())
85        {
86           try
87           {
88              ObjectInputStream input = new ObjectInputStream(
89                 new FileInputStream(slideshowFile));
90              slideshowList = (List<SlideshowInfo>) input.readObject();
91           }  // конец блока try
92           catch (final Exception e)
93           {
94              runOnUiThread(
95                 new Runnable()
96                 {
97                 public void run()
98                    {
99                       // сообщение об ошибке чтения
100                      Toast message = Toast.makeText(Slideshow.this,
101                         R.string.message_error_reading,
102                         Toast.LENGTH_LONG);
103                       message.setGravity(Gravity.CENTER,
104                          message.getXOffset() / 2,
105                          message.getYOffset() / 2);
106                       message.show(); // сообщение Toast
107                       Log.v(TAG, e.toString());
108                    }  // конец описания метода run
109                 }  // конец описания Runnable
110              );  // конец вызова runOnUiThread
111           }  // конец блока catch
112        }  // конец блока if
113
114        if (slideshowList == null) // если null, создание файла
115           slideshowList = new ArrayList<SlideshowInfo>();
116
117           return (Object) null; // метод должен соответствовать
                                    // типу возвращаемых данных
118        }  // конец описания метода doInBackground
```

продолжение

Листинг 13.8 *(продолжение)*

```

      // создание адаптера ListView в потоке GUI
      @Override
      protected void onPostExecute(Object result)
      {
         super.onPostExecute(result);

      // создание и настройка адаптера ListView
      slideshowAdapter =
         new SlideshowAdapter(Slideshow.this, slideshowList);
      slideshowListView.setAdapter(slideshowAdapter);
   }  // конец определения метода onPostEecute
}  // конец определения класса LoadSlideshowsTask

```

Подкласс SaveSlideshowsTask класса AsyncTask

С помощью метода doInBackground класса SaveSlideshowsTask (см. листинг 13.9) проверяется, существует ли файл EnhancedSlideshowData.ser (строка 143). Если файл не существует, выполняется его создание. Затем в строках 147–148 создается объект ObjectOutputStream. В строке 149 вызывается метод writeObject класса ObjectOutputStream, выполняющий запись объекта List<SlideshowInfo> в файл slideshowFile. В случае генерирования исключения (в строках 154–169) используется метод runOnUiThread класса для отображения объекта Toast из потока UI, сообщающего о сути проблемы.

Листинг 13.9. Класс SaveSlideshowsTask сериализирует объект List<SlideshowInfo> в файл

```
133   // Класс, выполняющий сохранение объекта
      // List<SlideshowInfo> на устройстве
134 private class SaveSlideshowsTask extends AsyncTask<Object,Object,Object>
135   {
136      // сохранение из потока, который не является потоком GUI
137      @Override
138      protected Object doInBackground(Object... arg0)
139      {
140         try
141         {
142            // если файл не существует, выполняется его создание
143            if (!slideshowFile.exists())
144               slideshowFile.createNewFile();
145
146            // создание объекта ObjectOutputStream, запись в него
               // slideshowList
147            ObjectOutputStream output = new ObjectOutputStream(
148               new FileOutputStream(slideshowFile));
149            output.writeObject(slideshowList);
150            output.close();
151         }  // конец блока try
```

```
        catch (final Exception e)
        {
           runOnUiThread(
           new Runnable()
           {
              public void run()
              {
                 // отображение сообщения об ошибке чтения
                 Toast message = Toast.makeText(Slideshow.this,
                    R.string.message_error_writing, Toast.LENGTH_LONG);
                 message.setGravity(Gravity.CENTER,
                    message.getXOffset() / 2,
                    message.getYOffset() / 2);
                 message.show(); // отображение сообщения Toast
                 Log.v(TAG, e.toString());
              }  // конец описания метода run
           }  // конец описания Runnable
        ); // конец вызова runOnUiThread
     }  // конец блока catch

     return (Object) null; // метод должен соответствовать
                           // типу возвращаемого результата
     } // конец описания метода doInBackground
  }  // конец описания класса SaveSlideshowsTask

```

Переопределение метода onActivityResult класса Activity

Метод onActivityResult (см. листинг 13.10) был изменен для сохранения объекта List<SlideshowInfo> после выхода пользователя из режима редактирования слайд-шоу. Для выполнения этой задачи в строке 251 создается объект подкласса AsyncTask класса SaveSlideshowsTask (см. листинг 13.9), а также вызывается на выполнение метод execute этого объекта.

Листинг 13.10. Переопределение методов onCreateOptionsMenu, onOptionsItemSelected и onActivityResult класса Activity

```
245  // обновляет компонент ListView после завершения редактирования
     // слайд-шоу
246  @Override
247  protected void onActivityResult(int requestCode, int resultCode,
248     Intent data)
249  {
250     super.onActivityResult(requestCode, resultCode, data);
251   new SaveSlideshowsTask().execute((Object[]) null); // сохранение
                                                        // слайд-шоу
252     slideshowAdapter.notifyDataSetChanged(); // обновление адаптера
253  }  // конец определения метода onActivityResult
254
```

Метод getThumbnail

Метод getThumbnail (см. листинг 13.11) обновлен и теперь поддерживает загрузку миниатюр для изображений и видеороликов (строки 439–453).

Листинг 13.11. Метод getThumbnail, обновленный для возврата миниатюры изображения или видеоролика

```
438  // метод утилиты, применяемый для получения объекта Bitmap,
     // содержащего миниатюру изображения
439  public static Bitmap getThumbnail(MediaItem.MediaType type, Uri uri,
440     ContentResolver cr, BitmapFactory.Options options)
441  {
442     Bitmap bitmap = null;
443     int id = Integer.parseInt(uri.getLastPathSegment());
444
445     if (type == MediaItem.MediaType.IMAGE) // если это изображение
446        bitmap = MediaStore.Images.Thumbnails.getThumbnail(cr, id,
447           MediaStore.Images.Thumbnails.MICRO_KIND, options);
448      else if (type == MediaItem.MediaType.VIDEO) // если это видео
449         bitmap = MediaStore.Video.Thumbnails.getThumbnail(cr, id,
450            MediaStore.Video.Thumbnails.MICRO_KIND, options);
451
452      return bitmap;
453  }  // конец определения метода getThumbnail
```

13.5.4. Класс SlideshowEditor

Класс SlideshowEditor (см. листинги 13.12–13.14), используемый в настоящем приложении, поддерживает создание фотоснимков и выбор видеороликов, включаемых в слайд-шоу. В этом разделе будут рассмотрены изменения, внесенные в этот класс для поддержки новых свойств.

Переопределение метода onActivityResult класса Activity

Класс SlideshowEditor содержит две дополнительные кнопки, используемые для выбора видеоролика и создания фотоснимка соответственно. Для поддержки этих функций в строки 71–72 были добавлены константы (см. листинг 13.12), передаваемые методу startActivityForResult класса Activity. Затем эти константы возвращаются методу onActivityResult для идентификации класса Activity, возвращающего результат. Метод onActivityResult был изменен таким образом, чтобы использовать эти константы для обработки ссылки Uri, возвращаемой для изображения или видеоролика.

Листинг 13.12. Обновленные константы и метод onActivityResult

```
68  // настройка идентификаторов ID для каждого типа
    // возвращаемого медиаресурса
69  private static final int PICTURE_ID = 1;
70  private static final int MUSIC_ID = 2;
71  private static final int VIDEO_ID = 3;
```

```
private static final int TAKE_PICTURE_ID = 4;

// вызывается, если возвращается Activity, запущенный
// из данного Activity
@Override
protected final void onActivityResult(int requestCode, int resultCode,
   Intent data)
{
if (resultCode == RESULT_OK) // если ошибки отсутствуют
{
   Uri selectedUri = data.getData();

   // если Activity возвращает изображение
   if (requestCode == PICTURE_ID ||
      requestCode == TAKE_PICTURE_ID || requestCode == VIDEO_ID )
      {
         // определение типа медиаресурса
         MediaItem.MediaType type = (requestCode == VIDEO_ID ?
            MediaItem.MediaType.VIDEO : MediaItem.MediaType.IMAGE);

          // добавление нового элемента MediaItem в слайд-шоу
          slideshow.addMediaItem(type, selectedUri.toString());

          // обновление ListView
          slideshowEditorAdapter.notifyDataSetChanged();
       }  // конец блока if
       else if (requestCode == MUSIC_ID) // Activity возвращает музыку
          slideshow.setMusicPath(selectedUri.toString());
       }  // конец блока if
    }  // конец определения метода onActivityResult
```

Слушатели событий для методов takePictureButton и addVideoButton

В листинге 13.13 представлены обработчики событий для методов takePictureButton (строки 128–141) и addVideoButton (строки 144–155). Чтобы выбрать видеоролик, слушатель addVideoButtonListener использует технику, описанную в листинге 12.18, но типу MIME присваивается значение "video/*". В результате пользователь может выбирать видеоролики, которые хранятся в памяти устройства.

Листинг 13.13. Слушатели событий для методов takePictureButton и addVideoButton

```
127   // вызывается, если пользователь касается кнопки "Take Picture"
128   private OnClickListener takePictureButtonListener =
129      new OnClickListener()
130   {
131      // запуск деятельности, выбирающей изображение
132      @Override
133      public void onClick(View v)
134      {
135         // создание нового объекта Intent, запускающего
            // Slideshowplayer Activity
```

продолжение ⇨

Листинг 13.13 *(продолжение)*

```
         Intent takePicture =
            new Intent(SlideshowEditor.this, PictureTaker.class);

         startActivityForResult(takePicture, TAKE_PICTURE_ID);
      }  // конец определения метода onClick
   };  // конец определения OnClickListener takePictureButtonListener

   // вызывается, если пользователь касается кнопки "Add Picture"
   private OnClickListener addVideoButtonListener = new OnClickListener()
   {
      // запуск деятельности, выбирающей изображение
      @Override
      public void onClick(View v)
      {
         Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
         intent.setType("video/*");
         startActivityForResult(Intent.createChooser(intent,
            getResources().getText(R.string.chooser_video)), VIDEO_ID);
      }  // конец определения метода onClick
   };  // конец определения OnClickListener addVideoButtonListener
```

Обновленный подкласс LoadThumbnailTask класса AsyncTask

В результате обновления класса LoadThumbnailTask (см. листинг 13.14) обеспечивается передача типа MediaItem методу getThumbnail класса Slideshow, который возвращает объект Bitmap миниатюры для выбранного изображения или видеоролика.

Листинг 13.14. Класс LoadThumbnailTask загружает миниатюры изображения или видеоролика в отдельном потоке

```
   // задача по загрузке миниатюр в отдельном потоке
  private class LoadThumbnailTask extends AsyncTask<Object,Object,Bitmap>
   {
      ImageView imageView; // отображение миниатюры

      // загрузка миниатюры: ImageView, MediaType
      // и Uri в качестве аргументов
      @Override
      protected Bitmap doInBackground(Object... params)
      {
         imageView = (ImageView) params[0];

         return Slideshow.getThumbnail((MediaItem.MediaType)params[1],
            (Uri) params[2], getContentResolver(),
            new BitmapFactory.Options());
      }  // конец определения метода doInBackground

      // настройка миниатюры в ListView
      @Override
      protected void onPostExecute(Bitmap result)
```

```
    {
        super.onPostExecute(result);
        imageView.setImageBitmap(result);
    } // конец определения метода onPostExecute
} // конец определения класса LoadThumbnailTask
```

13.5.5. Подкласс PictureTaker класса Activity

С помощью класса PictureTaker (см. листинги 13.15–13.20) можно создавать фотоснимки, которые добавляются в слайд-шоу. Чтобы создать фотоснимок, следует коснуться экрана устройства во время просмотра сюжета съемки.

Операторы package, import и переменная экземпляра класса SlideshowEditor

В листинге 13.15 начинается определение класса PictureTaker. Были выделены операторы import для новых классов и интерфейсов, рассмотренных в разделе 13. В строках 31–32 объявляется компонент SurfaceView, который отображает картинку предварительного просмотра камеры, а также объект SurfaceHolder, управляющий компонентом SurfaceView. В строке 35 объявляется объект Camera, обеспечивающий доступ к видеокамере устройства. В списке List<String> с именем effects (строка 36) хранятся поддерживаемые камерой цветовые эффекты. Этот список используется для заполнения меню, в котором пользователь выбирает применяемый к фотографии эффект (например, *black and white* (черно-белый), *sepia* (сепия) и т. д.). В списке List<Camera.Size> с именем sizes (строка 37) хранятся поддерживаемые размеры фотографий, просматриваемых на экране устройства. В текущем приложении используется первый набор поддерживаемых размеров. Если не выбран ни один цветовой эффект, переменной String effect присваивается значение константы EFFECT_NONE из Camera.Parameter.

Листинг 13.15. Операторы package, import и поля класса PictureTaker

```
 1  // PictureTaker.java
 2  // Класс Activity, используемый для фотосъемки с помощью
    // камеры устройства
 3  package com.deitel.enhancedslideshow;
 4
 5  import java.io.IOException;
 6  import java.io.OutputStream;
 7  import java.util.List;
 8
 9  import android.app.Activity;
10  import android.content.ContentValues;
11  import android.content.Intent;
12  import android.hardware.Camera;
13  import android.net.Uri;
14  import android.os.Bundle;
15  import android.provider.MediaStore.Images;
16  import android.util.Log;
17  import android.view.Gravity;
18  import android.view.Menu;
```

продолжение ⇪

Листинг 13.15 *(продолжение)*

```
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.Toast;

public class PictureTaker extends Activity
{
   private static final String TAG = "PICTURE_TAKER"; // для
                                                // регистрации ошибок

   private SurfaceView surfaceView; // предварительный просмотр камеры
   private SurfaceHolder surfaceHolder; // управлением изменениями
                                           // SurfaceView
   private boolean isPreviewing; // выполняется предварительный
                                 // просмотр?

   private Camera camera; // используется для захвата данных изображения
   private List<String> effects; // поддерживаемые эффекты цвета
                                 // для камеры
   private List<Camera.Size> sizes; // поддерживаемые размеры
                                    // просмотра для камеры
   private String effect = Camera.Parameters.EFFECT_NONE; // эффект,
                                                    //выбранный по умолчанию

```

Переопределение метода onCreate класса Activity

Метод onCreate (см. листинг 13.16) подготавливает представление, используемое для предварительного просмотра фотографии, подобное отображаемому на экране во время выполнения приложения Camera на реальном устройстве Android. В начале создается объект SurfaceView и регистрируется слушатель событий касания, возникающих после касания экрана пальцами. Затем объект PictureTaker Activity захватывает фотографию и сохраняет ее в галерее (gallery) устройства. Потом создается объект SurfaceHolder и регистрируется объект, применяемый для обработки Callback при создании, изменении или удалении управляемого объекта SurfaceView. Строка 56 требуется в версиях Android, предшествующих версии Android 3.0. Метод setType класса SurfaceHolder и соответствующий ему аргумент-константа устарели и игнорируются в Android 3.0 либо более новой версии.

Листинг 13.16. Переопределение метода onCreate из Activity в классе PictureTaker

```
40  // вызывается при первом создании деятельности
41  @Override
42  public void onCreate(Bundle bundle)
43  {
44     super.onCreate(bundle);
```

```
      setContentView(R.layout.camera_preview); // настройка разметки

      // иницализация surfaceView и настройка его слушателя прикосновений
      surfaceView = (SurfaceView) findViewById(R.id.cameraSurfaceView);
      surfaceView.setOnTouchListener(touchListener);

      // инициализация surfaceHolder и настройка объекта
      // для обработки соответствующих обратных вызовов
      surfaceHolder = surfaceView.getHolder();
      surfaceHolder.addCallback(surfaceCallback);

      // требуется для предварительного просмотра камеры
      // в операционных системах, предшествующих Android 3.0
      surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
   } // конец определения метода onCreate

```

Переопределение методов onCreateOptionsMenu и onOptionsItemSelected класса Activity

Метод onCreateOptionsMenu (см. листинг 13.17, строки 60–70) отображает в меню перечень поддерживаемых цветовых эффектов камеры. После выбора пользователем одного из параметров меню метод onOptionsItemSelected получает объект Camera.Parameter (строка 76), а затем использует метод setColorEffect этого объекта для настройки эффекта. В строке 78 используется метод setParameters камеры для изменения настроек камеры. Начиная с этого момента выбранный цветовой эффект применяется к картинке предварительного просмотра.

Листинг 13.17. Переопределение методов onCreateOptionsMenu и onOptionsItemSelected класса Activity

```
59   // создание меню Activity из списка поддерживаемых
     // цветовых эффектов
60   @Override
61   public boolean onCreateOptionsMenu(Menu menu)
62   {
63      super.onCreateOptionsMenu(menu);
64
65      // создание элементов меню для каждого поддерживаемого эффекта
66      for (String effect : effects)
67         menu.add(effect);
68
69         return true;
70   } // конец определения метода onCreateOptionsMenu
71
72   // обработка вариантов, выбранных в меню
73   @Override
74   public boolean onOptionsItemSelected(MenuItem item)
75   {
76     Camera.Parameters p = camera.getParameters(); // получение параметров
77      p.setColorEffect(item.getTitle().toString()); // установка
                                                      // эффекта цвета
```

продолжение ⇨

Листинг 13.17 *(продолжение)*

```
      camera.setParameters(p); // применение новых параметров
      return true;
   } // конец определения метода onOptionsItemSelected

```

Обработка методов Callbacks класса SurfaceHolder

После создания, изменения или уничтожения объекта SurfaceView вызываются соответствующие методы Callback класса SurfaceHolder. В листинге 13.18 представлен анонимный внутренний класс, реализующий SurfaceHolder.Callback.

Листинг 13.18. Анонимный внутренний класс, обрабатывающий события Surface.Callback

```
// обработка событий SurfaceHolder.Callback
private SurfaceHolder.Callback surfaceCallback =
   new SurfaceHolder.Callback()
{
   // освобождает ресурсы после разрушения SurfaceView
   @Override
   public void surfaceDestroyed(SurfaceHolder arg0)
   {
      camera.stopPreview(); // завершение просмотра Camera
      isPreviewing = false;
      camera.release(); // освобождение ресурсов объекта Camera
   } // конец определения метода surfaceDestroyed

   // инициализация камеры после создания SurfaceView
   @Override
   public void surfaceCreated(SurfaceHolder arg0)
   {
      // получение доступа к камере и к поддерживаемым эффектам
      // цвета/размерам предварительного просмотра
      camera = Camera.open(); // по умолчанию выбирается тыльная камера
      effects = camera.getParameters().getSupportedColorEffects();
      sizes = camera.getParameters().getSupportedPreviewSizes();
   } // конец описания метода surfaceCreated

   @Override
   public void surfaceChanged(SurfaceHolder holder, int format,
      int width, int height)
   {
      if (isPreviewing) // если уже есть предварительный просмотр
         camera.stopPreview(); // остановить просмотр

      // конфигурирование и настройка параметров камеры
      Camera.Parameters p = camera.getParameters();
      p.setPreviewSize(sizes.get(0).width, sizes.get(0).height);
      p.setColorEffect(effect); // использование выбранного эффекта
      camera.setParameters(p);  // применение новых параметров
```

```
117
118        try
119        {
120           camera.setPreviewDisplay(holder); // холдер, используемый
                                                 // для отображения
121        }  // конец блока try
122           catch (IOException e)
123        {
124           Log.v(TAG, e.toString());
125        }  // конец блока catch
126
127        camera.startPreview(); // начало просмотра
128        isPreviewing = true;
129     }  // конец определения метода surfaceChanged
130  }; // конец определения SurfaceHolder.Callback
131
```

Метод surfaceDestroyed (строки 88–93) класса SurfaceHolder.Callback завершает предварительный просмотр фотографий и освобождает ресурсы, используемые приложением Camera. С помощью метода surfaceCreated класса SurfaceHolder.Callback (строки 96–103) обеспечивается доступ к объекту Camera и к поддерживаемым этим объектом функциям. Статический объект open класса Camera получает доступ к объекту Camera, который обеспечивает использование приложением тыльной камеры устройства. Затем используется объект Parameters класса Camera для получения списка List<String>, представляющего эффекты, поддерживаемые камерой, и списка List<Camera.Size>, представляющего поддерживаемые размеры изображения предварительного просмотра.

ПРИМЕЧАНИЕ

В данном приложении не перехватывается возможный объект RuntimeException метода open, который генерируется в случае недоступности камеры.

Метод surfaceChanged (строки 105–129) интерфейса SurfaceHolder.Callback вызывается при каждом изменении размера или формата SurfaceView (обычно это происходит при вращении устройства и после первого создания и отображения SurfaceView). (В файле манифеста было отключено вращение для этого класса Activity.) В строке 109 проверяется, включен ли предварительный просмотр, и, в случае, если он включен, с помощью метода stopPreview класса Camera предварительный просмотр отключается. Затем мы получаем доступ к объекту Parameters класса Camera и вызываем метод setPreviewSize. Этот метод применяется для настройки размеров области предварительного просмотра камеры на основе значений ширины и высоты первого объекта (в списке List<Camera.Size> содержатся поддерживаемые значения размера области предварительного просмотра). В результате вызова метода setColorEffect для применения текущего эффекта цвета к области предварительного просмотра (и ко всем созданным фотоснимкам). Потом выполняется реконфигурирование объекта Camera путем вызова соответствующего метода setParameters, применяющего изменения. В строке 120 передается SurfaceHolder методу setPreviewDisplay класса Camera. Тем самым демонстрируется возможность отображения предварительного просмотра в SurfaceView. В строке 127 запускается предварительный просмотр с помощью метода startPreview класса Camera.

Обработка PictureCallbacks класса Camera

В листинге 13.19 определяется анонимный класс Camera.PictureCallback, который получает данные после создания фотоснимка пользователем. Метод onPictureTaken принимает массив типа byte, содержащий данные фотоснимка и объекта Camera, используемого при создании фотоснимка. В рассматриваемом примере байтовый массив imageData хранит версию фотоснимков в формате JPEG. Благодаря этому можно просто сохранить массив imageData в памяти устройства (строки 154–158). В строках 161–163 создается новый объект Intent, а его метод setData применяется для определения ссылки Uri для сохраненной фотографии в виде данных, возвращаемых из данного объекта Activity. Метод setResult (строка 163) класса Activity применяется для оповещения об отсутствии ошибок, а в качестве результата выбирается returnIntent. Объект Activity из SlideshowEditor может использовать данные объекта Intent для сохранения фотографии в слайд-шоу и загрузки соответствующей миниатюры.

Листинг 13.19. Реализация анонимного класса Camera.PictureCallback, используемого для сохранения фотоснимка

```
132   // обработка обратных вызовов объекта Camera
133   Camera.PictureCallback pictureCallback = new Camera.PictureCallback()
134   {
135      // вызывается при создании фотоснимка пользователем
136      public void onPictureTaken(byte[] imageData, Camera c)
137      {
138         // создание имени файла новой фотографии на основе
            // "Slideshow_" + текущее время (в миллисекундах)
139         String fileName = "Slideshow_" + System.currentTimeMillis();
140
141         // создание ContentValues и конфигурирование данных
            // нового фотоснимка
142         ContentValues values = new ContentValues();
143         values.put(Images.Media.TITLE, fileName);
144         values.put(Images.Media.DATE_ADDED, System.currentTimeMillis());
145         values.put(Images.Media.MIME_TYPE, "image/jpg");
146
147         // получение ссылки Uri для местоположения,
            // в котором сохраняются файлы
148         Uri uri = getContentResolver().insert(
149            Images.Media.EXTERNAL_CONTENT_URI, values);
150
151         try
152         {
153            // получение OutputStream для uri
154            OutputStream outStream =
155               getContentResolver().openOutputStream(uri);
156            outStream.write(imageData); // вывод изображения
157            outStream.flush();  // очистка буфера
158            outStream.close(); // закрытие потока
159
160            // Обект Intent, возвращающий данные SlideshowEditor
```

```
            Intent returnIntent = new Intent();
            returnIntent.setData(uri);          // возврат Uri
                                                // объекту SlideshowEditor
            setResult(RESULT_OK, returnIntent); // успешно
                                                // снята фотография

            // сообщение о сохранении фотоснимка
            Toast message = Toast.makeText(PictureTaker.this,
               R.string.message_saved, Toast.LENGTH_SHORT);
            message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
               message.getYOffset() / 2);
            message.show(); // отображение сообщения Toast

            finish(); // завершение и возврат к SlideshowEditor
         }  // конец блока try
         catch (IOException ex)
         {
            setResult(RESULT_CANCELED); // ошибка при фотосъемке

          // сообщение о сохранении фотоснимка
          Toast message = Toast.makeText(PictureTaker.this,
             R.string.message_error_saving, Toast.LENGTH_SHORT);
          message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
             message.getYOffset() / 2);
          message.show(); // отображение сообщения Toast
         } // конец блока catch
      }  // конец определения метода onPictureTaken
   };  // конец pictureCallback

```

Обработка событий касания для SurfaceView

Метод onTouch (см. листинг 13.20) создает фотоснимок после касания пользователем экрана. Метод takePicture класса Camera (строка 195) в асинхронном режиме создает фотографии с помощью камеры устройства. В качестве аргументов этот метод получает несколько слушателей. Первый из этих слушателей представляет собой экземпляр класса Camera.ShutterCallback, который уведомляется сразу же после создания фотоснимка. Этот экземпляр представляет собой уникальное средство обеспечения визуальной или звуковой обратной связи для процесса фотосъемки. В данном приложении эта обратная связь не используется, поэтому в качестве первого аргумента передается null. Последние два слушателя представляют собой экземпляры класса Camera.PictureCallback, обеспечивающего получение и обработку приложением данных изображения в формате RAW (несжатые данные изображения) и данных изображения в формате JPEG соответственно. В этом приложении не используются данные в формате RAW, поэтому второй аргумент takePicture также равен null. Третий обратный вызов использует pictureCallback (см. листинг 13.19) для обработки изображений в формате JPEG.

Листинг 13.20. Реализация метода OnTouchListener, обрабатывающего события касания

```
   // делает фотоснимок после касания экрана
   private OnTouchListener touchListener = new OnTouchListener()
   {
      @Override
      public boolean onTouch(View v, MotionEvent event)
      {
         // создание фотоснимка
         camera.takePicture(null, null, pictureCallback);
         return false;
      }  // конец определения метода onTouch
   }; // конец определения touchListener
} // конец определения класса PictureTaker
```

13.5.6. Класс SlideshowPlayer

Класс Activity SlideshowPlayer (см. листинги 13.21–13.23) воспроизводит слайд-шоу, сопровождаемое фоновой музыкой. В результате обновления объекта SlideshowPlayer стало возможным воспроизведение любых видеороликов, включенных в слайд-шоу. В этом разделе будут описаны только измененные фрагменты этого класса.

Операторы package, import и переменные экземпляра класса SlideshowEditor

В листинге 13.21 начинается определение класса SlideshowPlayer. Были выделены операторы import, соответствующие новым классам и интерфейсам, рассмотренным в разделе 13.3. Переменная videoView служит для управления компонентом VideoView, используемым для воспроизведения видеороликов.

Листинг 13.21. Операторы package, import и поля экземпляра класса SlideshowPlayer

```
// SlideshowPlayer.java
// Воспроизводит выбранное слайд-шоу, которое передается
// как дополнение объекта Intent
package com.deitel.enhancedslideshow;

import java.io.FileNotFoundException;
import java.io.InputStream;

import android.app.Activity;
import android.content.ContentResolver;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.TransitionDrawable;
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnCompletionListener;
```

```
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.widget.ImageView;
import android.widget.MediaController;
import android.widget.VideoView;

public class SlideshowPlayer extends Activity
{
   private static final String TAG = "SLIDESHOW";  // тег регистрации
                                                   // ошибок

   // константы, используемые для сохранения состояния слайд-шоу
   // при изменении конфигурации
   private static final String MEDIA_TIME = "MEDIA_TIME";
   private static final String IMAGE_INDEX = "IMAGE_INDEX";
   private static final String SLIDESHOW_NAME = "SLIDESHOW_NAME";

   private static final int DURATION = 5000; // 5 секунд на слайд
   private ImageView imageView;     // текущее изображение
   private VideoView videoView;     // текущий видеоролик
   private String slideshowName;    // имя текущего слайд-шоу
   private SlideshowInfo slideshow; // слайд-шоу воспроизводится
   private BitmapFactory.Options options; // параметры
                                          // загрузки изображений
   private Handler handler;  // используется для обновления слайд-шоу
   private int nextItemIndex; // индекс следующей отображаемой фотографии
   private int mediaTime; // время (в миллисекундах)
                          // воспроизведения медиаресурса
   private MediaPlayer mediaPlayer; // фоновая музыка (при наличии)

```

Переопределение метода onCreate класса Activity

В строках 55–65 содержатся изменения метода onCreate (см. листинг 13.22). В строке 55 получаем разметку VideoView, в строках 56–65 регистрируется слушатель OnCompletionListener, который служит для уведомления о завершении воспроизведения видеоролика в VideoView. Метод onCompletion вызывает метод postUpdate класса Handler и передает updateSlideshow Runnable в качестве аргумента для обработки следующего изображения или видеоролика в слайд-шоу.

Листинг 13.22. Переопределение метода onCreate класса Activity в классе SlideshowPlayer

```
47   // инициализирует SlideshowPlayer Activity
48   @Override
49   public void onCreate(Bundle savedInstanceState)
50   {
```

продолжение ⇗

Листинг 13.22 *(продолжение)*

```
      super.onCreate(savedInstanceState);
      setContentView(R.layout.slideshow_player);

      imageView = (ImageView) findViewById(R.id.imageView);
      videoView = (VideoView) findViewById(R.id.videoView);
      videoView.setOnCompletionListener( // настройка обработчика
                                         // завершения видео
      new OnCompletionListener()
      {
         @Override
         public void onCompletion(MediaPlayer mp)
         {
           handler.post(updateSlideshow); // обновление слайд-шоу
       }  // конец определения метода onCompletion
   }  // конец определения анонимного внутреннего класса
  ); // конец определения OnCompletionListener

  if (savedInstanceState == null) // запуск Activity
  {
     // получение имени слайд-шоу из дополнений к Intent
     slideshowName = getIntent().getStringExtra(Slideshow.NAME_EXTRA);
     mediaTime = 0;     // позиция в медиаклипе
     nextItemIndex = 0; // запуск первого изображения
   }  // конец блока if
   else // восстановление Activity
   {
      // получение позиции воспроизведения, сохраненной
      // при изменении конфигурации
      mediaTime = savedInstanceState.getInt(MEDIA_TIME);

      // получение индекса изображения, которое отображалось
      // при изменении конфигурации
      nextItemIndex = savedInstanceState.getInt(IMAGE_INDEX);

      // получение имени слайд-шоу, которое отображалось
      // при изменении конфигурации
      slideshowName = savedInstanceState.getString(SLIDESHOW_NAME);
   }  // конец блока else

   // получение SlideshowInfo для воспроизводимого слайд-шоу
   slideshow = Slideshow.getSlideshowInfo(slideshowName);

   // конфигурирование BitmapFactory.Options для загрузки изображений
    options = new BitmapFactory.Options();
    options.inSampleSize = 4; // выборка с 1/4 от исходной ширины/высоты

   // если есть музыка
   if (slideshow.getMusicPath() != null)
   {
      // попытка создания MediaPlayer для воспроизведения музыки
```

```
    try
    {
       mediaPlayer = new MediaPlayer();
       mediaPlayer.setDataSource(
          this, Uri.parse(slideshow.getMusicPath()));
       mediaPlayer.prepare(); // подготовка
                              // к воспроизведению с помощью MediaPlayer
       mediaPlayer.setLooping(true);  // циклическое
                                      // воспроизведение музыки
       mediaPlayer.seekTo(mediaTime); // установка на mediaTime
    }  // конец блока try
    catch (Exception e)
    {
       Log.v(TAG, e.toString());
    }  // конец блока catch
 }  // конец блока if

   handler = new Handler(); // создание обработчика для контроля
                            // слайд-шоу
 }  // конец описания метода onCreate
```

Изменения, внесенные в updateSlideshow Runnable

Объект updateSlideshow Runnable (см. листинг 13.23) обрабатывает изображения и видеоролики. Если слайд-шоу не завершено, с помощью метода run в строках 193–208 определятся, является ли следующий слайд в слайд-шоу изображением или видеороликом. Если этот слайд является изображением, в строках 197–198 отображается компонент ImageView, а компонент videoView скрывается. В строке 199 создается задача AsyncTask из LoadImageTask (определена в строках 213–249), выполняющая загрузку и отображение фотографии. Если же слайд является видеороликом, в строках 203–204 скрывается компонент ImageView и отображается компонент videoView, а в строке 205 вызывается метод playVideo (определен в строках 272–279). Метод playVideo воспроизводит видеофайл, находящийся по указанной ссылке Uri. В строке 275 вызывается метод setVideoUri класса VideoView, с помощью которого определяется местоположение воспроизводимого видеофайла. В строках 276–277 настраивается MediaController для компонента VideoView, используемый для отображения элементов управления воспроизведением видеоролика. В строке 278 начинается воспроизведение видеоролика с помощью метода start класса VideoView.

Листинг 13.23. Объект Runnable, выполняющий отображение фотографии или воспроизведение видеоролика

```
178   // анонимый внутренний класс, реализующий Runnable
      // для управления слайд-шоу
179   private Runnable updateSlideshow = new Runnable()
180   {
181      @Override
182      public void run()
183      {
```

продолжение ⇨

Листинг 13.23 *(продолжение)*

```
         if (nextItemIndex >= slideshow.size())
         {
            // если воспроизводится музыка
            if (mediaPlayer != null && mediaPlayer.isPlaying())
               mediaPlayer.reset(); // слайд-шоу завершено,
                                    // сброс MediaPlayer
               finish(); // возврат к запускающему  Activity
         }   // конец блока if
         else
         {
            MediaItem item = slideshow.getMediaItemAt(nextItemIndex);

         if (item.getType() == MediaItem.MediaType.IMAGE)
         {
            imageView.setVisibility(View.VISIBLE);//отображение ImageView
            videoView.setVisibility(View.INVISIBLE); // скрыть videoView
            new LoadImageTask().execute(Uri.parse(item.getPath()));
         }  // конец блока if
         else
         {
            imageView.setVisibility(View.INVISIBLE); // скрыть ImageView
            videoView.setVisibility(View.VISIBLE); // показать videoView
            playVideo(Uri.parse(item.getPath()));//воспроизведение видео
         }  // конец блока else

         ++nextItemIndex;
      }  // конец блока else
   }  // конец определения метода run

   // задача по загрузке миниатюр в отдельном потоке
   class LoadImageTask extends AsyncTask<Uri, Object, Bitmap>
   {
      // загрузка изображений
      @Override
      protected Bitmap doInBackground(Uri... params)
      {
         return getBitmap(params[0], getContentResolver(), options);
      }  // конец определения метода doInBackground

      // настройка миниатюры, отображаемой на ListView
      @Override
      protected void onPostExecute(Bitmap result)
      {
         super.onPostExecute(result);
         BitmapDrawable next = new BitmapDrawable(result);
         next.setGravity(android.view.Gravity.CENTER);
         Drawable previous = imageView.getDrawable();

         // если предыдущий TransitionDrawable,
         // получить его второй элемент Drawable
```

```
       if (previous instanceof TransitionDrawable)
          previous = ((TransitionDrawable) previous).getDrawable(1);

          if (previous == null)
             imageView.setImageDrawable(next);
          else
          {
             Drawable[] drawables = { previous, next };
             TransitionDrawable transition =
                new TransitionDrawable(drawables);
             imageView.setImageDrawable(transition);
             transition.startTransition(1000);
          }  // конец блока else

          handler.postDelayed(updateSlideshow, DURATION);
       }  // конец определения метода onPostExecute
    }  // конец определения класса LoadImageTask

    // метод утилиты, применяемый для получения Bitmap на основе Uri
    public Bitmap getBitmap(Uri uri, ContentResolver cr,
       BitmapFactory.Options options)
    {
       Bitmap bitmap = v;

       // получить изображение
       try
       {
          InputStream input = cr.openInputStream(uri);
          bitmap = BitmapFactory.decodeStream(input, null, options);
       }  // конец блока try
       catch (FileNotFoundException e)
       {
          Log.v(TAG, e.toString());
       }  // конец блока catch

       return bitmap;
    }  // конец определения метода getBitmap

       // воспроизведение видео
    private void playVideo(Uri videoUri)
    {
       // настройка и воспроизведение видеоролика
       videoView.setVideoURI(videoUri);
       videoView.setMediaController(
          new MediaController(SlideshowPlayer.this));
       videoView.start(); // запуск видео
    }  // конец определения метода playVideo
 };  // конец определения Runnable updateSlideshow
} // конец определения класса SlideshowPlayer
```

13.6. Резюме

При создании приложения использовались возможности сериализации объекта из пакета java.io для сохранения слайд-шоу на устройстве с целью дальнейшего просмотра. Для использования объекта с механизмом сериализации был реализован тегирующий интерфейс Serializable. С помощью метода writeObject класса ObjectOutputStream был создан объект graph и сериализованные объекты. Чтение и десериализация объектов осуществлялась с помощью метода readObject класса ObjectInputStream.

Была обеспечена возможность создания фотоснимков с помощью тыльной камеры устройства, сохранение полученных фотографий в Gallery устройства и добавление новой фотографии в слайд-шоу. Для выполнения этих задач использовался класс Camera. С помощью класса SurfaceView осуществлялся предварительный просмотр фотографии. При касании пользователем экрана класс Camera получал указание сделать фотографию, а объект Camera.PictureCallback извещался о том, что фотография сделана и обработаны данные фотографии. Также использовались поддерживаемые классом Camera эффекты цвета.

Приложение Slideshow использовало Intent для запуска Activity, с помощью которого осуществлялся выбор изображений из Gallery. Аналогичная методика использовалась для выбора пользователем видеороликов, но для каждого типа данных указывались различные типы MIME, в результате чего отображались только видеоролики.

С помощью компонента VideoView воспроизводились видеоролики в слайд-шоу. Для выполнения воспроизведения указывались ссылка URI видеоролика, отображаемого VideoView, и MediaController. С помощью слушателя MediaPlayer.OnCompletionListener определяется момент завершения воспроизведения видеоролика.

В следующей главе будут рассмотрены важные методики, используемые при разработке приложений планшета на основе Android 3.x. Также мы познакомимся с веб-службами WeatherBug для создания приложения Weather Viewer.

Приложение Weather Viewer 14

Веб-службы, документы JSON, фрагменты, ListFragment, DialogFragment, ActionBar, навигационная панель с вкладками, виджеты, объекты Broadcast Intents и BroadcastReceivers

В этой главе...

- Использование веб-служб WeatherBug® для получения сведений о текущей погоде и пятидневного прогноза для выбранного города, а также обработка данных с помощью Android 3.x JsonReader.
- Создание повторно используемых компонентов с помощью фрагментов, а также оптимизация использования экрана приложениями планшета.
- Реализация навигации с вкладками с помощью Android 3.x ActionBar.
- Создание вспомогательного виджета приложения, который может устанавливаться на начальном экране пользователя.
- Трансляция изменений для выбранного в приложении города во вспомогательный виджет приложения.

14.1. Введение

В приложении Weather Viewer app (рис. 14.1) с помощью веб-служб WeatherBug® обеспечивается получение данных о текущем состоянии погоды и пятидневный прогноз



Рис. 14.1. Приложение Weather Viewer, отображающее погоду в Бостоне, штат Массачусетс

для выбранного города. Это приложение включает предварительно созданный список городов причем после установки приложения по умолчанию выбран `Boston`.

В этой главе будет разработано планшетное приложение Android, основанное на использовании преимуществ различных новых свойств, появившихся в Android 3.x. С помощью объекта Android 3.x `JsonReader` будут считываться данные о погоде, возвращаемые веб-службами `WeatherBug`. В результате данные будут передаваться приложению в формате JSON (*JavaScript Object Notation*, Запись объекта JavaScript).

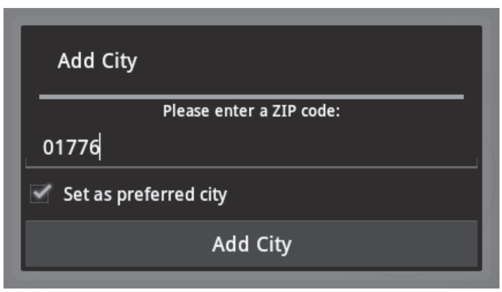


Рис. 14.2. Диалоговое окно Add City, в котором введен почтовый индекс и установлен флажок Set as preferred city

Мы используем панель действий Android 3.x, находящуюся в верхней части экрана, для размещения меню и других навигационных элементов приложения. Чтобы добавить новый город, коснитесь пункта меню Add New City (Добавить новый город), находящегося на панели действий. На экране отобразится диалоговое окно (рис. 14.2), в котором вводится почтовый индекс и выбирается другой город. Можно также переключиться между текущим и пятидневным прогнозом (рис. 14.3) с помощью навигационной *панели с вкладками* (Current Conditions либо Five Day Forecast, находящейся справа от названия приложения на рис. 14.1).

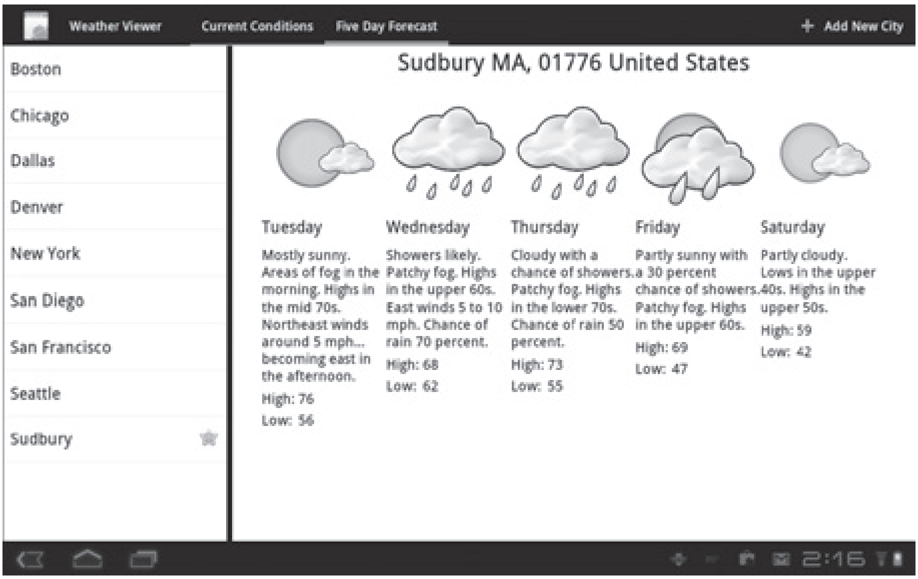


Рис. 14.3. Приложение Weather Viewer отображает пятидневный прогноз погоды для города Sudbury, штат Массачусетс

Список городов, текущая погода, пятидневный прогноз и диалоговые окна, применяемые в приложении, были реализованы с помощью фрагментов Android 3.x, представляющих собой повторно используемые компоненты пользовательского интерфейса Activity. Чтобы воспользоваться преимуществами больших экранов планшетов, класс Activity может одновременно отображать несколько фрагментов. Список городов отображается в виде ListFragment — объект Fragment, включающий компонент ListView. После длительного нажатия на название города в списке городов отображается диалоговое окно DialogFragment, в котором можно удалить выбранный город либо выбрать его в качестве предпочтительного (для этого города отображается текущая погода после первой загрузки приложения). Диалоговое окно, отображаемое после выбора параметра меню Add New City на панели действий, — тоже DialogFragment. После касания названия города отображается информация о погоде в этом городе с помощью объекта Fragment.

Это приложение также включает вспомогательный виджет (ис. 14.4), который устанавливается на одном из начальных экранов. Виджеты появились еще в ранних версиях Android. В Android 3.x появилась возможность *изменения размеров* окон

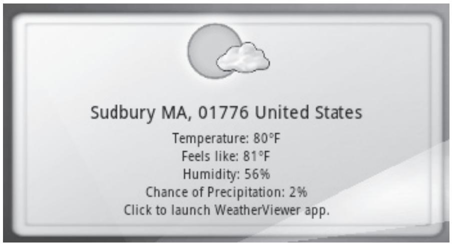


Рис. 14.4. Вспомогательный виджет приложения Weather Viewer, отображающий текущую погода для выбранного города

виджетов. С помощью виджета Weather Viewer можно просматривать текущую погоду для выбранного города на начальном экране.

14.2. Тестирование приложения Weather Viewer

Открытие и выполнение приложения

Откройте среду Eclipse и импортируйте проект Weather Viewer app. Выполните следующие действия:

1. Выполните команды File ▸ Import... (Файл ▸ Импорт...) для открытия диалогового окна Import (Импорт).
2. В диалоговом окне Import раскройте узел General (Общий) и выберите параметр Existing Projects into Workspace (Существующие проекты в рабочую область). Потом щелкните на кнопке Next > (Далее >).
3. Справа от текстового поля Select root directory (Выбрать корневой каталог) щелкните на кнопке Browse... (Просмотр...). Затем найдите и выделите папку WeatherViewer.
4. Щелкните на кнопке Finish (Готово) для выполнения импорта проекта.

Приложение получает сведения о погоде от веб-служб WeatherBug. Для выполнения созданного в этой главе приложения зарегистрируйте свой собственный ключ WeatherBug API на веб-сайте weather.weatherbug.com/desktop-weather/api.html.

Как только будет получен ключ API, используйте его для замены YOUR_API_KEY в строке 62 класса ReadLocationTask, в строке 66 класса ReadForecastTask, в строке 53 класса ReadFiveDayForecastTask. Введя ключ API, щелкните правой кнопкой мыши на проекте приложения, находящемся в окне Package Explorer, затем в появившемся меню выберите команды Run As ▸ Android Application (Выполнить как ▸ Приложение Android).

Просмотр текущей погоды и пятидневного прогноза для выбранного города

Для просмотра текущей погоды коснитесь названия города в списке городов; для просмотра пятидневного прогноза коснитесь значка Five Day Forecast (Пятидневный прогноз), находящегося на панели действий. Поверните планшет, чтобы перейти из портретной ориентации в альбомную (или наоборот) и увидеть различия макетов

экрана для каждой ориентации. Чтобы вернуться к просмотру текущей погоды, коснитесь значка Current Conditions (Текущая погода), находящегося на панели действий.

Добавление города

Чтобы добавить город в список, коснитесь значка Add New City (Добавить город), расположенного на панели действий. На экране появится диалоговое окно Add City (Добавить город). Введите почтовый индекс (ZIP code) для добавляемого города. Если нужно выбрать город в качестве предпочтительного, установите флажок Set as preferred city (Выбрать предпочтительный город). Коснитесь кнопки Add City (Добавить город), чтобы добавить город в список.

Удаление города из списка городов и изменение предпочтительного города

Чтобы удалить город из списка городов или изменить предпочтительный город, нажмите название города и не отпускайте палец до тех пор, пока не отобразится диалоговое окно с тремя кнопками, — Set as Preferred City (Выбрать в качестве предпочтительного города), Delete (Удалить) и Cancel (Отмена). Затем коснитесь кнопки, соответствующей выполняемой задаче. Если будет удален предпочтительный город, в качестве предпочтительный автоматически выбирается первый город в списке.

Добавление виджета приложения на начальный экран

Чтобы добавить на начальный экран виджет, связанный с данным приложением, коснитесь кнопки home устройства, затем нажмите и удерживайте палец на пустой области начального экрана для отображения списка виджетов, которые могут быть установлены. Выполняйте прокрутку вправо, пока не найдете виджет Weather Viewer. Коснитесь виджета, чтобы добавить его на выбранный начальный экран, либо перетащите виджет на один из пяти начальных экранов. Как только вы добавите виджет на начальный экран, он автоматически отобразит текущую погоду для предпочтительного города. Чтобы удалить виджет, нажмите на его значок и, не отпуская палец, перетащите значок на кнопку Remove (Удалить), расположенную в правом верхнем углу экрана. Чтобы изменить размеры виджета, нажимайте на его значок в течение нескольких секунд, а затем уберите палец с экрана. Операционная система Android отобразит маркеры изменения размера, которые можно использовать для изменения размеров виджета.

14.3. Обзор применяемых технологий

Android 3.x Fragment, ListFragment and DialogFragment

Фрагменты — это новое ключевое свойство, которое появилось в Android 3.x. С помощью *фрагментов* представляются повторно используемые части пользовательского интерфейса Activity и код логики, используемой в приложении. Разрабатываемое в главе приложение использует фрагменты для создания и управления компонентами графического интерфейса пользователя. Путем комбинирования нескольких фрагментов создаются устойчивые пользовательские интерфейсы, которые сполна используют

преимущества большого экрана планшета. Фрагменты могут также «меняться местами», придавая тем самым больший динамизм графическому интерфейсу пользователя.

Базовый класс для всех фрагментов — Fragment (пакет android.app). В этом приложении используются фрагменты различных типов. Фрагмент, включающий компонент ListView, отображается в виде ListFragment. Для отображения диалоговых окон применяются объекты DialogFragments. Текущая погода и пятидневный прогноз отображаются с помощью подклассов класса Fragment.

Несмотря на то что фрагменты появились в Android 3.x, их можно использовать в более ранних версиях Android с помощью пакета обеспечения совместимости. Последнюю версию этого пакета можно загрузить с веб-сайта http://developer.android.com/sdk/compatibility-library.html.

Управление фрагментами

Подобно Activity, каждый Fragment имеет свой *жизненный цикл*. Методы жизненного цикла Fragment будут рассматриваться по мере их использования. Объекты Fragments должны храниться в родительском классе Activity, поскольку они не могут выполняться независимо. Для объектов Fragments приложения в качестве родительского класса Activity используется главный класс Activity приложения под названием WeatherViewerActivity. Для управления фрагментами в родительском классе Activity применяется FragmentManager (пакет android.app). С помощью объекта FragmentTransaction (пакет android.app), полученного из FragmentManager, класс Activity может добавлять, удалять и выполнять переходы между фрагментами.

Разметки фрагментов

Подобно Activity, каждый Fragment имеет собственную разметку, которая обычно определяется как ресурс XML-разметки, хотя и может быть создана в динамическом режиме. Для объектов Fragment, используемых для формирования пятидневного прогноза, используются различные макеты для портретной и альбомной ориентации, что позволяет лучше использовать большой экран планшета приложения. Пятидневный прогноз отображается в направлении слева направо при выборе альбомной ориентации и в направлении сверху вниз в случае выбора портретной ориентации. Для идентификации текущей ориентации используется объект Configuration класса Activity (пакет android.content.res). После определения ориентации выбирается используемая разметка.

Панель действий Android 3.x

В Android 3.x появилась панель действий, отображаемая в верхней части экрана, которая заменила строку заголовка приложения, используемую в предыдущих версиях Android. В левой части панели отображаются пиктограмма и название приложения. Также на панели действий могут отображаться меню параметров приложения, элементы навигации (панель навигации с вкладками) и другие интерактивные компоненты GUI. В данном приложении панель действий применяется для реализации панели навигации с вкладками, в которой можно выбрать объект Fragment, отображающий текущую погоду или пятидневный прогноз погоды для заданного города. В приложении также имеется меню параметров, в котором можно выбрать параметр для добавления нового города в объект ListFragment, применяемый для хранения списка городов. Можно также

определить элементы меню в качестве действий, которые могут быть помещены на панель действий (при наличии свободного места). Для этого нужно установить атрибут android:showAsAction, соответствующий пункту меню.

Обработка длительных нажатий

Для выполнения обработки длительных нажатий элементов списка городов из фрагмента ListFragment применяется слушатель AdapterView.OnItemLongClickListener (пакет android.widget), который реагирует на это событие, обеспечивая выбор предпочтительного города, удаление города или отмену этой операции.

Вспомогательный виджет приложения

В этом приложении используется вспомогательный виджет приложения, отображающий текущую погоду для выбранного пользователем с помощью приложения Weather Viewer предпочтительного города. Чтобы выбрать и добавить виджет, нужно применить жест длительного нажатия на начальном экране. Для создания виджета приложения и обеспечения приема системных уведомлений в случае активизации, отключения, удаления или обновления виджета приложения был расширен класс AppWidgetProvider (пакет android.appwidget), который является подклассом класса BroadcastReceiver (пакет android.content).

Объект PendingIntent, используемый для запуска Activity из виджета приложения

Общепринятой практикой запуска приложений на выполнение является касание виджета приложения, отображаемого на начальном экране устройства. Объект PendingIntent (пакет android.app) позволяет запустить приложение и просмотреть текущий прогноз для предпочтительного города.

Объект JsonReader и веб-службы

В этом приложении используется объект JsonReader (пакет android.util), применяемый для чтения содержимого объектов JSON, содержащих сведения о погоде. С помощью объекта URL определяется URL-ссылка, с помощью которой вызывается веб-служба WeatherBug RESTful, возвращающая объекты JSON. По указанной URL-ссылке открывается поток InputStream, в котором вызывается веб-служба. Объект JsonReader получает данные из этого потока InputStream.

Трансляция объектов Intents и Receivers

Вспомогательный виджет приложения Weather Viewer отображает текущую погоду для предпочтительного города, выбранного в приложении. Этот город может быть изменен пользователем в любой момент времени. Как только происходит изменение, приложение с помощью объекта Intent транслирует его. Виджет приложения использует объект BroadcastReceiver (пакет android.content) для прослушивания изменения, после чего он может отобразить текущую погоду для выбранного города.

14.4. Создание графического интерфейса пользователя и файлов ресурсов приложения

В этом разделе будут рассмотрены новые свойства графического интерфейса пользователя и файлы ресурсов для приложения Weather Viewer. В целях экономии места мы не приводим код файла ресурсов strings.xml, но зато приводим код большинства файлов разметки XML.

14.4.1. Файл AndroidManifest.xml

В листинге 14.1 приводится код файла AndroidManifest.xml для этого приложения. Для подключения библиотеки Android 3.1 SDK атрибуту android:minSdkVersion элемента uses-sdk присваивается значение "12" (строка 5). Это приложение может выполняться только на устройствах Android 3.1+ и виртуальных устройствах AVD. В строках 6–7 указывается, что это приложение требует подключения к Интернету. Элемент receiver (строки 19–30) регистрируется класс WeatherProvider (который представляет виджет приложения) в качестве BroadcastReceiver, определяет XML-файл для метаданных виджета приложения, а также определяет фильтры объекта Intent класса WeatherProvider. В строке 32 в качестве службы регистрируется вложенный класс WeatherService класса WeatherProvider, который может запускаться для выполнения в фоновом режиме. Класс WeatherService применяется для обновления данных о погоде в виджете приложения. Подобно действиям все службы должны регистрироваться в файле манифеста, иначе они не смогут выполняться.

Листинг 14.1. Файл AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android=http://schemas.android.com/apk/res/android
3     package="com.deitel.weatherviewer" android:versionCode="1"
4     android:versionName="1.0">
5     <uses-sdk android:minSdkVersion="12" />
6     <uses-permission android:name="android.permission.INTERNET">
7     </uses-permission>
8
9     <application android:icon="@drawable/icon"
10       android:label="@string/app_name">
11       <activity android:name=".WeatherViewerActivity"
12          android:label="@string/app_name">
13          <intent-filter>
14             <action android:name="android.intent.action.MAIN" />
15             <category android:name="android.intent.category.LAUNCHER" />
16          </intent-filter>
17       </activity>
18
19       <receiver android:name=".WeatherProvider">
20          <meta-data android:name="android.appwidget.provider"
21             android:resource="@xml/weather_widget_provider_info" />
```

```
         <intent-filter>
            <action android:name=
               "android.appwidget.action.APPWIDGET_UPDATE" />
         </intent-filter>
         <intent-filter>
            <action android:name=
               "com.deitel.weatherviewer.UPDATE_WIDGET" />
          </intent-filter>
      </receiver>

      <service android:name=".WeatherProvider$WeatherService" />
   </application>
</manifest>
```

14.4.2. Разметка класса WeatherViewerActivity, определенная в файле main.xml

В файле ресурса main.xml (листинг 14.2) определяется разметка класса WeatherViewer-Activity. С помощью элемента fragment включается объект CitiesFragment в качестве первого потомка корневого объекта LinearLayout. Объект CitiesFragment создается автоматически после «раздувания» разметки с помощью WeatherViewerActivity. В качестве метки-заполнителя, отображающй фрагменты ForecastFragments, используется объект forecast_replacer FrameLayout. С помощью этой метки-заполнителя определяются размеры и местоположение области в Activity, в которой будут отображаться фрагменты ForecastFragments. Класс WeatherViewerActivity осуществляет перестановку фрагментов ForecastFragments с помощью методов FragmentTransactions.

Листинг 14.2. Файл разметки main.xml класса WeatherViewerActivity

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
   android:orientation="horizontal" android:layout_width="match_parent"
   android:layout_height="match_parent">
   <fragment class="com.deitel.weatherviewer.CitiesFragment"
      android:id="@+id/cities" android:layout_weight="3"
      android:layout_width="wrap_content"
      android:layout_height="match_parent"/>
   <FrameLayout android:layout_width="8dp"
      android:layout_height="match_parent"
      android:background="@android:color/black"/>
   <FrameLayout android:id="@+id/forecast_replacer"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
   android:layout_weight="1" android:background="@android:color/white"/>
</LinearLayout>
```

14.4.3. Использование файла arrays.xml для хранения заданных по умолчанию городов и почтовых индексов

Заданные по умолчанию города и соответствующие им почтовые индексы хранятся в файле ресурса приложения arrays.xml (листинг 14.3). Это обеспечивает возможность непосредственного считывания значений ресурсов String (в отличие от чтения каждого элемента отдельно). Два массива String загружаются в WeatherViewerActivity путем вызова метода getStringArray класса Resources.

Листинг 14.3. Использование массива arrays.xml для хранения заданных по умолчанию городов и почтовых индексов

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string-array name="default_city_names">
      <item>Boston</item>
      <item>Chicago</item>
      <item>Dallas</item>
      <item>Denver</item>
      <item>New York</item>
      <item>San Diego</item>
      <item>San Francisco</item>
      <item>Seattle</item>
   </string-array>
   <string-array name="default_city_zipcodes">
      <item>02115</item>
      <item>60611</item>
      <item>75254</item>
      <item>80202</item>
      <item>10024</item>
      <item>92104</item>
      <item>94112</item>
      <item>98101</item>
   </string-array>
</resources>
```

14.4.4. Разметка меню WeatherViewerActivity, определенная в файле actionmenu.xml

В файле ресурса actionmenu.xml (листинг 14.4) определяются элементы меню ActionBar. Причем атрибуты ресурса меню будут теми же, что и для стандартных меню Android. Мы представим вашему вниманию новый атрибут android:showAsAction, определяющий, каким образом элемент меню отображается на панели ActionBar. С помощью значения ifRoom этого атрибута определяется отображение элемента меню на панели ActionBar при наличии достаточного места на самой панели. Если же присвоить атрибуту значение always, элемент меню будет постоянно отображаться на панели ActionBar, но при этом возникнет риск перекрытия соседних элементов меню. Использование значения withText для атрибута android:title приведет к отображению значения типа String вместе с элементом меню.

Листинг 14.4. Разметка меню actionmenu.xml для класса WeatherViewerActivity

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/add_city_item"
      android:icon="@android:drawable/ic_input_add"
      android:title="@string/add_new_city"
      android:showAsAction="ifRoom|withText"/>
</menu>
```

14.4.5. Разметка и конфигурирование виджета приложения с помощью файла WeatherProvider

В файле weather_widget_provider_info.xml (листинг 14.5) определяются метаданные для объекта AppWidgetProvider класса WeatherViewer. С помощью атрибутов minWidth и minHeight определяются начальные размеры виджета приложения. В результате обеспечивается одновременное изменение размеров находящихся на начальном экране значков и виджетов. Начальный экран устройства Android делится на одинаковые по размеру ячейки, как описано на веб-сайте http://developer.android.com/guide/practices/ui_guidelines/widget_design.html#sizes.

С помощью атрибутов minWidth и minHeight выбираются различные стандартные размеры виджета. Ресурс разметки виджета приложения определяется с помощью атрибута initialLayout. Атрибут updatePeriodMillis определяет частоту, с которой объект AppWidgetProvider должен получать широковещательный объект Intent под названием ACTION_APPWIDGET_UPDATE. При каждом получении объекта Intent класс WeatherProvider (раздел 14.5.11) запускает новую службу WeatherService, с помощью которой выполняется обновление сведений о текущей погоде виджета приложения. При этом игнорируются значения этого атрибута, не превышающие 30 минут. Если виджеты приложений требуют более частого обновления, они обращаются к AlarmManager. Атрибут android:resizeMode появился в Android 3.1 и служит для определения направлений, в которых виджет приложения должен изменять размеры на начальном экране.

Листинг 14.5. Конфигурирование виджета приложения WeatherProvider

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
   xmlns:android=http://schemas.android.com/apk/res/android
   android:minWidth="212dp" android:minHeight="148dp"
   android:initialLayout="@layout/weather_app_widget_layout"
   android:updatePeriodMillis="3600000"
   android:resizeMode="horizontal|vertical"/>
```

Разметка виджета приложения определяется в файле weather_app_widget_layout.xml, в котором используется простой вложенный макет LinearLayout. В качестве основного фона LinearLayout выбирается один из стандартных наборов границ виджета приложения от Google, который можно загрузить с веб-сайта http://developer.android.com/guide/practices/ui_guidelines/widget_design.html#frames.

14.5. Создание приложения

Это приложение состоит из 11 классов, которые подробно рассматриваются в разделах 14.5.1–14.5.11. Ниже представлен краткий обзор этих классов и связей между ними.

- Класс WeatherViewerActivity (раздел 14.5.1) является единственным классом Activity приложения. Класс Activity использует фрагмент AddCityDialogFragment (раздел 14.5.3) для добавления новых городов в приложение. Класс Activity включает лишь один экземпляр класса CitiesFragment (раздел 14.5.2), который всегда располагается у левого края экрана. Класс WeatherViewerActivity выполняет перестановку различных фрагментов ForecastFragments (разделы 14.5.4, 14.5.5 и 14.5.8), которые отображаются в правой части экрана приложения. Этот класс Activity также содержит код панели ActionBar и загружает заданные по умолчанию города и города, которые пользователь добавил в приложение.
- Класс ReadLocationTask (раздел 14.5.6) получает информацию о местонахождении почтового индекса (ZIP code) с помощью веб-служб WeatherBug. Эта информация используется в WeatherViewerActivity, двумя подклассами ForecastFragment и виджетом приложения.
- Класс SingleForecastFragment (раздел 14.5.5) — это объект Fragment, который отображает однодневный прогноз погоды. Отображаемые данные считываются с помощью обьекта AsyncTask ReadForecastTask (раздел 14.5.7).
- Класс FiveDayForecastFragment (раздел 14.5.8) подобен классу SingleForecastFragment, отличается тем, что отображает пятидневный прогноз, полученный от AsyncTask ReadFiveDayForecastTask (раздел 14.5.9). Класс DailyForecast (раздел 14.5.10) представляет прогноз погоды на один день. С помощью этого класса упрощается передача информации обратно, от задачи ReadFiveDayForecast.
- Класс WeatherProvider (раздел 14.5.11) управляет и обновляет виджет приложения. В дополнение к стандартным широковещательным сообщениям, принимаемым виджетом из системы, виджет получает широковещательные сообщения от WeatherViewerActivity в случае изменения предпочтительного города.

14.5.1. Класс WeatherViewerActivity

Класс WeatherViewerActivity (листинг 14.6) включает несколько новых операторов import (новые свойства выделены маркером). Этот класс реализует интерфейс DialogFinishedListener (определен в листинге 14.29), что позволяет ему обрабатывать добавление пользователем нового города. Поля класса будут рассматриваться на протяжении этого раздела (по мере их использования).

Листинг 14.6. Операторы package, import и поля экземпляра класса WeatherViewerActivity

```
// WeatherViewerActivity.java
// Главный класс Activity для приложения Weather Viewer.
package com.deitel.weatherviewer;

```

```
import java.util.HashMap;
import java.util.Map;

import android.app.ActionBar;
import android.app.ActionBar.Tab;
import android.app.ActionBar.TabListener;
import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.os.Handler;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;

import com.deitel.weatherviewer.AddCityDialogFragment.
   DialogFinishedListener;
import com.deitel.weatherviewer.CitiesFragment.CitiesListChangeListener;
import com.deitel.weatherviewer.ReadLocationTask.LocationLoadedListener;

public class WeatherViewerActivity extends Activity implements
DialogFinishedListener
{
   public static final String WIDGET_UPDATE_BROADCAST_ACTION =
      "com.deitel.weatherviewer.UPDATE_WIDGET";

   private static final int BROADCAST_DELAY = 10000;

   private static final int CURRENT_CONDITIONS_TAB = 0;

   public static final String PREFERRED_CITY_NAME_KEY =
      "preferred_city_name";
   public static final String PREFERRED_CITY_ZIPCODE_KEY =
      "preferred_city_zipcode";
   public static final String SHARED_PREFERENCES_NAME =
      "weather_viewer_shared_preferences";
   private static final String CURRENT_TAB_KEY = "current_tab";
   private static final String LAST_SELECTED_KEY = "last_selected";

   private int currentTab; // местоположение текущей выделенной вкладки
   private String lastSelectedCity; // последний город, выбранный
                                    // в списке
   private SharedPreferences weatherSharedPreferences;

   // сохраняет названия городов и соответствующие почтовые индексы
   private Map<String, String> favoriteCitiesMap;
```

продолжение ⇗

Листинг 14.6 *(продолжение)*

```
   private CitiesFragment listCitiesFragment;
   private Handler weatherHandler;

```

Метод onCreate класса WeatherViewerActivity

Метод onCreate (листинг 14.7) инициализирует новый класс WeatherViewerActivity. Вызывается метод getFragmentManager класса Activity (строка 66), с помощью которого открывается доступ к фрагменту FragmentManager, используемому для взаимодействия с фрагментами текущего класса Activity. В данном случае мы получаем доступ к фрагменту CitiesFragment. Для любого фрагмента класса Activity также доступен FragmentManager. В дополнение к инициализации нескольких других переменных экземпляра класса вызывается метод setupTabs (определен в листинге 14.22) для инициализации объекта ActionBar класса Activity.

Листинг 14.7. Переопределение метода onCreate класса WeatherViewerActivity

```
57 // инициализирует этот класс Activity и «раздувает» /
   // разметку из xml-файла
58 @Override
59 public void onCreate(Bundle savedInstanceState)
60 {
61    super.onCreate(savedInstanceState); // передает данный
                                         // Bundle суперклассу
62    setContentView(R.layout.main);      //«раздувает» разметку из main.xml
63
64    // получает фрагмент из CitiesFragment
65    listCitiesFragment = (CitiesFragment)
66       getFragmentManager().findFragmentById(R.id.cities);
67
68    // настройка  CitiesListChangeListener
69    listCitiesFragment.setCitiesListChangeListener(
70       citiesListChangeListener);
71
72    // создание HashMap для хранения названий городов
      // и соответствующих почтовых индексов
73    favoriteCitiesMap = new HashMap<String, String>();
74
75    weatherHandler = new Handler();
76
77    weatherSharedPreferences = getSharedPreferences(
78       SHARED_PREFERENCES_NAME, MODE_PRIVATE);
79
80    setupTabs(); // настройка навигационных вкладок ActionBar
81 }  // конец определения метода onCreate
82
```

Методы onSaveInstanceState и onRestoreInstanceState класса WeatherViewerActivity

Метод onSaveInstanceState (листинг 14.8, строки 84–92) сохраняет позицию текущей выделенной вкладки и выбранного элемента списка. Индекс текущей выделенной папки добавляется в выбранный объект Bundle с помощью метода putInt класса Bundle. Эти значения считываются с помощью метода onRestoreInstanceState (строки 95–104), который обеспечивает возможность отображения классом Activity одного и того же города и выделенной вкладки после изменения ориентации устройства.

Листинг 14.8. Переопределение методов onSaveInstanceState и onRestoreInstanceState в классе WeatherViewerActivity

```
 83 // сохранение состояния класса Activity
 84 @Override
 85 public void onSaveInstanceState(Bundle savedInstanceStateBundle)
 86 {
 87    // сохранение текущей выбранной вкладки
 88    savedInstanceStateBundle.putInt(CURRENT_TAB_KEY, currentTab);
 89    savedInstanceStateBundle.putString(LAST_SELECTED_KEY,
 90       lastSelectedCity); // сохранение текущего выбранного города
 91       super.onSaveInstanceState(savedInstanceStateBundle);
 92 }  // конец описания метода onSaveInstanceState
 93
 94 // восстановление сохраненного состояния класса Activity
 95 @Override
 96 public void onRestoreInstanceState(Bundle savedInstanceStateBundle)
 97 {
 98    super.onRestoreInstanceState(savedInstanceStateBundle);
 99
100    // получение выделенной вкладки
101    currentTab = savedInstanceStateBundle.getInt(CURRENT_TAB_KEY);
102    lastSelectedCity = savedInstanceStateBundle.getString(
103       LAST_SELECTED_KEY); // получение выбранного города
104 }  // конец определения метода onRestoreInstanceState
105
```

Метод onResume класса WeatherViewerActivity

С помощью метода onResume класса Activity осуществляется заполнение списка избранных городов (листинг 14.9). Если favoriteCitiesMap пуст, считывается список сохраненных городов из объекта SharedPreferences приложения путем вызова метода loadSavedCities (листинг 14.13). Если данные в SharedPreferences отсутствуют, favoriteCitiesMap останется пустым. В этом случае вызывается метод addSampleCities (листинг 14.18), который добавляет заранее сконфигурированный список городов, хранящийся в XML-ресурсах. Мы указываем текущую выбранную вкладку объекта ActionBar с помощью соответствующего метода selectTab (строка 124), а затем загружаем прогноз погоды для выбранного города путем вызова метода loadSelectedForecast (листинг 14.11).

Листинг 14.9. Переопределение метода onResume класса WeatherViewerActivity

```
// вызывается при восстановлении этого класса Activity
@Override
public void onResume()
{
   super.onResume();

   if (favoriteCitiesMap.isEmpty()) // если список городов пуст
   {
      loadSavedCities(); // загрузка ранее добавленных городов
   }  // конец блока if

   // если здесь не осталось городов
   if (favoriteCitiesMap.isEmpty())
   {
      addSampleCities(); // добавить выборку городов
   }  // конец блока if

   // загрузка ранее выбранного прогноза
   getActionBar().selectTab(getActionBar().getTabAt(currentTab));
   loadSelectedForecast();
}  // конец определения метода onResume

```

Реализация интерфейса CitiesListChangeListener

Слушатель CitiesListChangeListener (листинг 14.10) получает обновления из класса CitiesFragment, как только пользователь выбирает новый город или изменяет предпочтительный город. Метод onSelectedCityChanged (строки 133–138) вызывается после выбора пользователем нового города. Имя текущего города передается методу selectForecast класса WeatherViewerActivity (листинг 14.16) для отображения прогноза, соответствующего выбранному городу, с помощью фрагмента ForecastFragment. Если предпочтительный город был изменен, соответствующее сообщение передается методу onPreferredCityChanged (строки 141–146). Мы передаем название текущего города методу setPreferred класса WeatherViewerActivity (листинг. 14.12) для обновления содержимого объекта SharedPreferences приложения.

Листинг 14.10. Реализация интефейса CitiesListChangeListener

```
128   // слушает изменения в CitiesFragment
129   private CitiesListChangeListener citiesListChangeListener =
130   new CitiesListChangeListener()
131   {
132      // вызывается при изменении выбранного города
133      @Override
134      public void onSelectedCityChanged(String cityNameString)
135      {
136         // отображение прогноза для выбранного города
137         selectForecast(cityNameString);
```

```
    }  // конец определения метода onSelectedCityChanged

    // вызывается при изменении предпочтительного города
    @Override
    public void onPreferredCityChanged(String cityNameString)
    {
       // сохранение нового предпочтительного города
       // в SharedPreferences приложения
       setPreferred(cityNameString);
    }  // конец определения метода onPreferredCityChanged
 }; // конец определения CitiesListChangeListener

```

Метод loadSelectedForecast класса WeatherViewerActivity

Метод loadSelectedForecast (листинг 14.11) вызывает метод selectForecast (листинг 14.16) для загрузки прогноза для последнего города, который был выбран пользователем в фрагменте CitiesFragment. Если город не выбран, загружается прогноз для предпочтительного города.

Листинг 14.11. Метод loadSelectedForecast класса WeatherViewerActivity

```
 // загрузка ранее выбранного прогноза
 private void loadSelectedForecast()
 {
    // если есть ранее сохраненный город
    if (lastSelectedCity != null)
    {
       selectForecast(lastSelectedCity); // последний выбранный город
    }  // конец блока if
    else
    {
       // получение имени предпочтительного города
       String cityNameString = weatherSharedPreferences.getString(
          PREFERRED_CITY_NAME_KEY, getResources().getString(
             R.string.default_zipcode));
      selectForecast(cityNameString); // загрузка прогноза для
                                      // предпочтительного города
    } // конец блока else
 }  // конец определения loadSelectedForecast

```

Метод setPreferred класса WeatherViewerActivity

Метод setPreferred (листинг 14.12) обновляет запись, соответствующую предпочтительному городу, объекта SharedPreferences. Мы получаем почтовый индекс путем сравнения имени данного города, отображаем окно Editor с помощью метода edit класса SharedPreferences. Имя и почтовый индекс нового предпочтительного города передаются методу putString класса Editor. Для сохранения изменений используется метод apply класса SharedPreferences. Мы удаляем последний выделенный город и загружаем класс loadSelectedForecast (см. листинг 14.11) для отображения прогноза погоды для

нового предпочтительного города. Затем создается объект Intent типа WIDGET_UPDATE_BROADCAST_ACTION, который распространяется с помощью метода sendBroadcast класса Activity. Если пользователь установил виджет приложения на начальном экране, объект WeatherProvider (раздел 14.5.11) будет получать эту трансляцию и обновлять виджет приложения, отображая прогноз погоды для нового предпочтительного города. Многие веб-службы, включая поддерживаемые WeatherBug, ограничивают количество и частоту вызовов службы. Исходя из этих соображений, используется объект Handler, который отсылает уведомление после краткой паузы. Благодаря этому предотвращается одновременный вызов веб-службы приложением и виджетом приложения для загрузки нового прогноза погоды.

Листинг 14.12. Метод setPreferred класса WeatherViewerActivity

```
// настройка предпочтительного города
public void setPreferred(String cityNameString)
{
   // получение почтового индекса города
   String cityZipcodeString = favoriteCitiesMap.get(cityNameString);
   Editor preferredCityEditor = weatherSharedPreferences.edit();
   preferredCityEditor.putString(PREFERRED_CITY_NAME_KEY,
      cityNameString);
   preferredCityEditor.putString(PREFERRED_CITY_ZIPCODE_KEY,
      cityZipcodeString);
   preferredCityEditor.apply(); // подтверждение изменений
   lastSelectedCity = null; // удаление последнего выбранного прогноза
   loadSelectedForecast();  // загрузка прогноза для
                            // предпочтительного города

   // обновление виджета приложения для отображения нового
   // предпочтительного города
   final Intent updateWidgetIntent = new Intent(
      WIDGET_UPDATE_BROADCAST_ACTION);

   // отправка широковещательного уведомления после краткой паузы
   weatherHandler.postDelayed(new Runnable()
   {
      @Override
      public void run()
      {
         sendBroadcast(updateWidgetIntent); // распространение интента
      }
   }, BROADCAST_DELAY);
}  // конец определения метода setPreferred

```

Метод loadSavedCities класса WeatherViewerActivity

Метод loadSavedCities (листинг 14.13) загружает список избранных городов из объекта SharedPreferences приложения. Пара, включающая карту каждого города и почтовый индекс, выбирается с помощью метода getAll класса SharedPreferences. Перебор этих пар

с последующим их добавлением в список выполняется с помощью метода addCity класса WeatherViewerActivity (листинг 14.14).

Листинг 14.13. Метод loadSavedCities класса WeatherViewerActivity

```
// считывает ранее сохраненный список городов из объекта
// SharedPreferences
private void loadSavedCities()
{
   Map<String, ?> citiesMap = weatherSharedPreferences.getAll();

   for (String cityString : citiesMap.keySet())
   {
      // если это значение не является предпочтительным городом
      if (!(cityString.equals(PREFERRED_CITY_NAME_KEY) ||
         cityString.equals(PREFERRED_CITY_ZIPCODE_KEY)))
         {
            addCity(cityString, (String) citiesMap.get(cityString),
               false);
      }  // конец блока if
   }  // конец цикла for
}  // конец описания метода loadSavedCities

```

Метод addSampleCities класса WeatherViewerActivity

Метод addSampleCities (см. листинг 14.14) считывает заданные по умолчанию избранные города из файла ресурса приложения arrays.xml. С помощью метода getStringArray класса Resource (строки 216–217 и 220–221) выбираются массивы, содержащие заданные по умолчанию названия городов и почтовые индексы. С помощью метода addCity выполняется обход каждого города и добавление его в список (листинг 14.15). Название первого города в выборке передается методу setPreferred класса WeatherViewerActivity для выбора этого города в качестве предпочтительного (см. листинг 14.12).

Листинг 14.14. Метод addSampleCities класса WeatherViewerActivity

```
212   // добавление городов в выборку
213   private void addSampleCities()
214   {
215      // загрузка массива названий городов из ресурсов
216      String[] sampleCityNamesArray = getResources().getStringArray(
217         R.array.default_city_names);
218
219      // загрузка массива почтовых индексов из ресурсов
220      String[] sampleCityZipcodesArray = getResources().getStringArray(
221         R.array.default_city_zipcodes);
222
223      // для каждого города из выборки
224      for (int i = 0; i < sampleCityNamesArray.length; i++)
225      {
```

продолжение ☞

Листинг 14.14 *(продолжение)*

```
         // выбор первого города из выборки в качестве
         // предпочтительного по умолчанию
         if (i == 0)
         {
            setPreferred(sampleCityNamesArray[i]);
         }  // конец блока if

         // добавление города в список
         addCity(sampleCityNamesArray[i], sampleCityZipcodesArray[i],
            false);
      }  // конец цикла for
   } // конец описания метода addSampleCities

```

Метод addCity класса WeatherViewerActivity

Добавление новых городов во фрагмент CitiesFragment (раздел 14.5.2) осуществляется с помощью метода addCity (листинг 14.15). Выбранная пара «название города-почтовый индекс» добавляется в фрагмент favoriteCitiesMap, затем передается методу addCity класса CitiesFragment. Город также добавляется в объект приложения SharedPreferences, а затем вызывается метод apply для сохранения нового города.

Листинг 14.15. Метод addCity класса WeatherViewerActivity

```
// добавление нового города во фрагменты CitiesFragment ListFragment
public void addCity(String city, String zipcode, boolean select)
{
   favoriteCitiesMap.put(city, zipcode);     // добавить
                                             // в HashMap of cities
   listCitiesFragment.addCity(city, select); // добавить город
                                             // в объект Fragment
   Editor preferenceEditor = weatherSharedPreferences.edit();
   preferenceEditor.putString(city, zipcode);
   preferenceEditor.apply();
}  // конец описания метода addCity

```

Метод selectForecast класса WeatherViewerActivity

Метод selectForecast (листинг 14.16) отображает сведения о прогнозе погоды для выбранного города. С помощью метода findFragmentById класса Fragment получаем доступ к текущему фрагменту, отображающему прогноз. Этому методу передается ID компонента FrameLayout из разметки Activity. После первого выполнения этого метода возвращается null. Диспетчер FragmentManager может получать доступ к фрагменту Fragment, отображающему прогноз, после замены FrameLayout фрагментом Fragment во время выполнения FragmentTransaction. Если в качестве выбранной вкладки ActionBar выступает вкладка Current Conditions, создается новый фрагмент ForecastFragment на основе данного почтового индекса (строки 270–271). В противном случае должна быть выбрана вкладка Five Day Forecast, в результате чего создается новый фрагмент FiveDayForecastFragment (строки

276–277). С помощью метода beginTransaction класса FragmentManager создается новый объект FragmentTransaction (строки 281–282). Объекты FragmentTransaction применяются для добавления, удаления и замены фрагментов Fragments наравне с другими действиями. В данном случае заменяется фрагмент Fragment, находящийся в правой части окна Activity, только что созданным фрагментом Fragment. Методу setTransition нового фрагмента передается константа TRANSIT_FRAGMENT_FADE класса FragmentTransaction (строки 285–286), в результате чего создается эффект визуального затухания при переходе от старого фрагмента к новому. Потом вызывается метод replace класса ForecastFragment (строки 290–291), в качестве аргументов которого используются ID заменяемого элемента и новый объект Fragment. Метод commit класса FragmentTransaction (строка 293) начинает выполнение транзакции.

Листинг 14.16. Метод selectForecast класса WeatherViewerActivity

```
248   // отображение прогноза для выбранного города
249   public void selectForecast(String name)
250   {
251      lastSelectedCity = name; // сохранение названия города
252      String zipcodeString = favoriteCitiesMap.get(name);
253      if (zipcodeString == null) // если почтовый индекс не найден
254      {
255         return; // не пытайтесь загрузить фрагмент
256      }  // конец блока if
257
258      // получение текущего отображаемого ForecastFragment
259      ForecastFragment currentForecastFragment = (ForecastFragment)
260      getFragmentManager().findFragmentById(R.id.forecast_replacer);
261
262      if (currentForecastFragment == null ||
263         !(currentForecastFragment.getZipcode().equals(zipcodeString) &&
264         correctTab(currentForecastFragment)))
265      {
266         // если выбрана вкладка "Current Conditions"
267         if (currentTab == CURRENT_CONDITIONS_TAB)
268         {
269         // создание нового ForecastFragment на основе почтового индекса
270            currentForecastFragment = SingleForecastFragment.newInstance(
271               zipcodeString);
272         }  // конец блока if
273         else
274         {
275         // создание нового ForecastFragment на основе почтового индекса
276           currentForecastFragment = FiveDayForecastFragment.newInstance(
277               zipcodeString);
278         }  // конец блока else
279
280         // создание нового FragmentTransaction
281         FragmentTransaction forecastFragmentTransaction =
282            getFragmentManager().beginTransaction();
283
```

продолжение ⇨

Листинг 14.16 *(продолжение)*

```
         // настройка эффекта затухания перехода
         forecastFragmentTransaction.setTransition(
            FragmentTransaction.TRANSIT_FRAGMENT_FADE);

         // замена Fragment (или View) с данным id новым
         // фрагментом
         forecastFragmentTransaction.replace(R.id.forecast_replacer,
            currentForecastFragment);

         forecastFragmentTransaction.commit(); // начало перехода
      } // конец блока if
   } // конец определения метода selectForecast

```

Методы correctTab и selectTab класса WeatherViewerActivity

Метод correctTab (листинг 14.17, строки 298–313) возвращает значение true, если данный объект ForecastFragment соответствует выбранной в настоящий момент времени вкладке (в частности, если выбрана вкладка Current Conditions и рассматривается объект SingleForecastFragment, либо выбрана вкладка Five Day Forecast и рассматривается объект FiveDayForecastFragment). На основе полученной таким образом информации метод selectForecast определяет необходимость обновления отображаемого фрагмента ForecastFragment. Метод selectTab (строки 316–320) выбирает вкладку с данным индексом. Этот индекс сохраняется в переменной экземпляра класса currentTab, а затем вызывается метод loadSelectedForecast (см. листинг 14.11).

Листинг 14.17. Методы correctTab и selectTab класса WeatherViewerActivity

```
297   // подходит ли этот ForecastFragment для текущей выбранной вкладки?
298   private boolean correctTab(ForecastFragment forecastFragment)
299   {
300   // если выбрана вкладка "Current Conditions"
301   if (currentTab == CURRENT_CONDITIONS_TAB)
302   {
303      // возврашает true, если данный объект ForecastFragment
304      // является объектом SingleForecastFragment
305      return (forecastFragment instanceof SingleForecastFragment);
306   } // конец блока if
307      else // выбрана вкладка "Five Day Forecast"
308      {
309         // возвращает true, если данный объект ForecastFragment
310         // является объектом FiveDayForecastFragment
311         return (forecastFragment instanceof FiveDayForecastFragment);
312      } // конец блока else
313   } // конец определения метода correctTab
314
315   // выбор вкладки в данной позиции
316   private void selectTab(int position)
317   {
```

```
    currentTab = position; // сохранение позиции вкладки
    loadSelectedForecast();
 } // конец определения метода selectTab

```

Переопределение методов onCreateOptionsMenu и onOptionsItemSelected класса Activity

Метод onCreateOptionsMenu (листинг 14.18, строки 323–332) инициализирует кнопку Add New City на панели ActionBar. С помощью метода getMenuInflator класса Activity получаем глобальный объект MenuInflator. С помощью этого объекта «раздуваем» меню, определенное в файле разметки actionmenu.xml, и связываем его с данным объектом Menu. Метод onOptionsItemSelected (строки 335–346) вызывается в том случае, если пользователь выбирает элемент Add New City на панели ActionBar. Затем подтверждаем, что MenuItem соответствует ожидаемому ID ресурса, вызываем метод showAddCityDialog (листинг 14.19) для отображения AddCityDialogFragment (раздел 14.5.3). Затем возвращается значение true, свидетельствующее о том, что выбранный элемент меню был обработан с помощью этого метода.

Листинг 14.18. Переопределение методов onCreateOptionsMenu и onOptionsItemSelected класса Activity

```
322   // создает меню Activities
323   @Override
324   public boolean onCreateOptionsMenu(Menu menu)
325   {
326      super.onCreateOptionsMenu(menu);
327      MenuInflater inflater = getMenuInflater();//глобальный MenuInflator
328
329      // «раздувание» разметки, определенной в файле actionmenu.xml
330      inflater.inflate(R.menu.actionmenu, menu);
331      return true; // возвращает true после создания меню
332   } // конец определения метода onCreateOptionsMenu
333
334   // если произведен щелчок на одном из элементов
335   @Override
336   public boolean onOptionsItemSelected(MenuItem item)
337   {
338      // если выбран элемент "Add City"
339      If (item.getItemId() == R.id.add_city_item)
340      {
341         showAddCityDialog(); // показать диалоговое окно
                                 // для ввода данных пользователем
342         return true; // возвращает true после обработки выделения
343      } // конец блока if
344
345      return false; // не обработаны неожидаемые элементы меню
346   } // конец определения метода onOptionsItemSelected
347
```

Методы showAddCityDialog и onDialogFinished класса WeatherViewerActivity

Метод showAddCityDialog (см. листинг 14.19, строки 349–364) отображает фрагмент DialogFragment, с помощью которого пользователь может вводить почтовый индекс. После создания нового фрагмента AddCityDialogFragment мы получаем FragmentManager класса Activity (строка 356). Для создания нового объекта FragmentTransaction используется метод beginTransaction класса FragmentManager. Объект FragmentTransaction передается методу show класса DialogFragment для отображения этого объекта поверх Activity. Также можно внедрить FragmentDialog в иерархию компонентов View класса Activity (здесь не показано). Метод onDialogFinished (строки 367–372) вызывается после исчезновения диалогового окна AddCityDialog. Аргумент zipcodeString представляет почтовый индекс, введенный пользователем. Аргумент preferred типа boolean имеет значение true, если пользователь установил флажок Set as preferred city. Оба эти аргумента передаются методу getCityNameFromZipcode (листинг 14.20).

Листинг 14.19. Методы showAddCityDialog и onDialogFinished класса WeatherViewerActivity

```
348   // отображает FragmentDialog, обеспечивающее возможность
      // добавления пользователем нового города
349   private void showAddCityDialog()
350   {
351      // создание нового фрагмента AddCityDialogFragment
352      AddCityDialogFragment newAddCityDialogFragment =
353         new AddCityDialogFragment();
354
355      // получение экземпляра класса FragmentManager
356      FragmentManager thisFragmentManager = getFragmentManager();
357
358      // начало FragmentTransaction
359      FragmentTransaction addCityFragmentTransition =
360         thisFragmentManager.beginTransaction();
361
362      // отображает окно DialogFragment
363      newAddCityDialogFragment.show(addCityFragmentTransition, "");
364   } // конец определения метода showAddCityDialog
365
366   // вызывается, если исчезает окно FragmentDialog
367   @Override
368   public void onDialogFinished(String zipcodeString, boolean preferred)
369   {
370      // преобразует почтовый индекс в город
371      getCityNameFromZipcode(zipcodeString, preferred);
372   } // конец определения метода onDialogFinished
373
```

Метод getCityNameFromZipcode

Метод getCityNameFromZipcode (см. листинг 14.20) запускает новый ReadLocationTask (раздел 14.5.6) для выборки названия города по заданному почтовому индексу. Если почтовый индекс уже находится в списке избранных городов, вместо запуска AsyncTask отображается сообщение Toast, в котором говорится о том, что пользователь не может добавлять дубликаты городов.

Листинг 14.20. Метод getCityNameFromZipcode класса WeatherViewerActivity

```
// считывание названия города из почтового индекса
private void getCityNameFromZipcode(String zipcodeString,
   boolean preferred)
{
   // если этот почтовый индекс уже добавлен
   if (favoriteCitiesMap.containsValue(zipcodeString))
   {
      // создание объекта Toast, отображающего сведения об ошибках
      Toast errorToast = Toast.makeText(WeatherViewerActivity.this,
         WeatherViewerActivity.this.getResources().getString(
         R.string.duplicate_zipcode_error), Toast.LENGTH_LONG);
      errorToast.setGravity(Gravity.CENTER, 0, 0);
      errorToast.show(); // отображение объекта Toast
   }  // конец блока if
   else
   {
      // загрузка информации о местоположении в фоновом потоке
      new ReadLocationTask(zipcodeString, this,
        new CityNameLocationLoadedListener(zipcodeString, preferred)).
      execute();
   }  // конец блока else
}  // конец определения метода getCityNameFromZipcode

```

Реализация интерфейса LocationLoadedListener

Интерфейс CityNameLocationLoadedListener (листинг 14.21) принимает информацию от завершенной задачи ReadLocationTask. После создания интерфейса LocationLoadedListener можно определить, является ли данное местоположение предпочтительным городом, воспользовавшись параметром preferred, имеющим тип boolean. Чтобы добавить город в список избранных городов, название города и почтовый индекс передаются методу addCity класса WeatherViewerActivity. Третий аргумент этого метода определяет загрузку прогноза погоды для нового города. Если новый город выбран в качестве предпочтительного, название города передается методу setPreferred.

Листинг 14.21. Реализация интерфейса LocationLoadedListener

```
// прослушивание информации о городе, загруженной в фоновую задачу
private class CityNameLocationLoadedListener implements
   LocationLoadedListener
```

продолжение ⇗

Листинг 14.21 *(продолжение)*

```
      {
         private String zipcodeString; // «разыскиваемый» почтовый индекс
         private boolean preferred;

         // создание нового интерфейса CityNameLocationLoadedListener
         public CityNameLocationLoadedListener(String zipcodeString,
            boolean preferred)
         {
            this.zipcodeString = zipcodeString;
            this.preferred = preferred;
        }  // конец определения интерфейса CityNameLocationLoadedListener

        @Override
      public void onLocationLoaded(String cityString, String stateString,
        String countryString)
        {
           // если найден город с таким же почтовым индексом
           if (cityString != null)
           {
              addCity(cityString, zipcodeString, !preferred);
                                               // добавление нового города

           if (preferred) // если это местоположение является
                          // предпочтительным городом
           {
              // сохранение предпочтительного города в SharedPreferences
              setPreferred(cityString);
           }  // конец блока if
        }  // конец блока if
        else
        {
           // отображение текста, сообщающего о невозможности поиска
           // информации
         Toast zipcodeToast = Toast.makeText(WeatherViewerActivity.this,
            WeatherViewerActivity.this.getResources().getString(
            R.string.invalid_zipcode_error), Toast.LENGTH_LONG);
            zipcodeToast.setGravity(Gravity.CENTER, 0, 0);
            zipcodeToast.show(); // отображение Toast
        }  // конец блока else
      }  // конец определения метода onLocationLoaded
   }  // конец определения класса CityNameLocationLoadedListener

```

Метод setupTabs класса WeatherViewerActivity

Навигационная панель с вкладками класса ActionBar инициализируется с помощью метода setupTabs (листинг 14.22). Путем вызова метода getActionBar класса Activity получаем ссылку на объект ActionBar. Объект ActionBar используется вместо строки заголовка во всех приложениях для третьей версии Android и предлагает функции и возможности, позволяющие осуществлять навигацию в приложении с помощью

вкладок и раскрывающихся меню. Затем передается константа NAVIGATION_MODE_TABS из объекта ActionBar соответствующему методу setNavigationMode, в результате чего сообщается об использовании вкладок (объекты Tabs). С помощью метода newTab класса ActionBar создаются два объекта Tab (строки 449 и460), обеспечивающие возможность выбора между текущей погодой и пятидневным прогнозом погоды. Для каждого объекта Tab настраивается текст и регистрируется соответствующий TabListener (weatherTabListener, определенный в листинге 14.19). В строках 457 и 464 добавляются объекты Tabs в ActionBar с помощью объекта addTab класса ActionBar. Как упоминалось выше, создаются два объекта Tabs, Current Conditions и Five Day Forecast.

Листинг 14.22. Метод setupTabs класса WeatherViewerActivity

```
440  // настройка вкладок из ActionBar
441  private void setupTabs()
442  {
443     ActionBar weatherActionBar = getActionBar(); // получить ActionBar
444
445     // выбор режима навигации ActionBar, использующего вкладки
446     weatherActionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
447
448     // создание вкладок "Current Conditions"
449     Tab currentConditionsTab = weatherActionBar.newTab();
450
451     // настройка заголовка Tab
452     currentConditionsTab.setText(getResources().getString(
453        R.string.current_conditions));
454
455     // настройка слушателя объекта Tab
456     currentConditionsTab.setTabListener(weatherTabListener);
457    weatherActionBar.addTab(currentConditionsTab); // добавление вкладки
458
459     // создание вкладки "Five Day Forecast"
460     Tab fiveDayForecastTab = weatherActionBar.newTab();
461     fiveDayForecastTab.setText(getResources().getString(
462        R.string.five_day_forecast));
463     fiveDayForecastTab.setTabListener(weatherTabListener);
464     weatherActionBar.addTab(fiveDayForecastTab);
465
466     // выбор вкладки "Current Conditions" по умолчанию
467     currentTab = CURRENT_CONDITIONS_TAB;
468  }  // конец определения метода setupTabs
469
```

Реализация интерфейса TabListener

В листинге 14.23 реализован интерфейс TabListener, который выполняет обработку событий, происходящих в случае выбора пользователем вкладок, созданных в листинге 14.21. Метод onTabSelected (строки 480–485) вызывает функцию selectTab (см. листинг 14.17) с выделенным индексом объекта Tab для отображения соответствующих сведений о погоде.

Листинг 14.23. Реализация интерфейса TabListener

```
// прослушивание событий, сгенерированных объектами Tabs из ActionBar
TabListener weatherTabListener = new TabListener()
{
   // вызывается, если выделенная вкладка выделяется повторно
   @Override
   public void onTabReselected(Tab arg0, FragmentTransaction arg1)
   {
   }  // конец определения метода onTabReselected

   // вызывается, если повторно выделяется ранее выбранный объект Tab
   @Override
   public void onTabSelected(Tab tab, FragmentTransaction arg1)
   {
     // отображает информацию, соответствующую выделенному объекту Tab
       selectTab(tab.getPosition());
   }  // конец определения метода onTabSelected

   // вызывается, если отменяется выделение вкладки
   @Override
   public void onTabUnselected(Tab arg0, FragmentTransaction arg1)
   {
   }  // конец определения метода onTabSelected
}; // конец определения WeatherTabListener
}  // конец определения класса WeatherViewerActivity
```

14.5.2. Класс CitiesFragment

Класс CitiesFragment определяет объект ListFragment, предназначенный для хранения списка городов. Иерархия объектов View класса WeatherViewerActivity включает единственный объект CitiesFragment, который остается закрепленным в левой части окна Activity все время.

Операторы package, import класса CitiesFragment, поля и вложенный интерфейс CitiesListChangeListener

В листинге 14.24 начинается определение класса CitiesFragment. Этот объект Fragment отправляет отчет о взаимодействии с пользователем родительскому классу Activity, который реализует вложенный интерфейс CitiesListChangeListener (строки 40–47; реализован в листинге 14.10). Метод onSelectedCityChanged вызывается в том случае, если пользователь выбирает название города в списке городов. Метод onPreferredCityChanged создает отчет об изменениях для предпочтительного города.

Листинг 14.24. Операторы package, import класса CitiesFragment, поля и вложенный интерфейс CitiesListChangeListener

```
1   // CitiesFragment.java
2   // Объект Fragment, отображающий список избранных городов.
3   package com.deitel.weatherviewer;
```

```
 4
 5 import java.util.ArrayList;
 6 import java.util.List;
 7
 8 import android.app.AlertDialog;
 9 Import android.app.ListFragment;
10 import android.content.Context;
11 import android.content.DialogInterface;
12 import android.content.SharedPreferences;
13 import android.content.SharedPreferences.Editor;
14 import android.content.res.Resources;
15 import android.graphics.Color;
16 import android.os.Bundle;
17 import android.view.Gravity;
18 import android.view.View;
19 import android.view.ViewGroup;
20 import android.widget.AdapterView;
21 import android.widget.AdapterView.OnItemLongClickListener;
22 import android.widget.ArrayAdapter;
23 import android.widget.ListView;
24 import android.widget.TextView;
25 import android.widget.Toast;
26
27 public class CitiesFragment extends ListFragment
28 {
29    private int currentCityIndex; // выделенная позиция текущего списка
30
31    // ключ, использованный для сохранения выбранного списка в Bundle
32    private static final String CURRENT_CITY_KEY = "current_city";
33
34   public ArrayList<String> citiesArrayList; // перечень названий городов
35    private CitiesListChangeListener citiesListChangeListener;
36   private ArrayAdapter<String> citiesArrayAdapter;
37
38   // интерфейс, описывающий слушателя изменений для выбранного
39   // города и предпочтительного города
40   public interface CitiesListChangeListener
41   {
42      // выбранный город изменен
43      public void onSelectedCityChanged(String cityNameString);
44
45      // предпочтительный город изменен
46      public void onPreferredCityChanged(String cityNameString);
47  }   // конец описания интерфейса CitiesListChangeListener
48
```

Методы onActivityCreated и setCitiesListChangeListener класса CitiesFragment

Метод onActivityCreated (листинг 14.25, строки 50–78) инициализирует компонент ListView данного ListFragment. Сначала выполняется проверка, не имеет ли данный объект

Bundle значение null. Если результаты проверки отрицательны, осуществляется выборка выделенного города с помощью метода getInt класса Bundle. В результате обеспечивается постоянство выбранного элемента списка при изменениях ориентации. Затем, используя контекст деятельности, элемент list из разметки в файле city_list_item.xml и пустой список ArrayList, мы создаем новый объект ListAdapter, имеющий тип CitiesArrayAdapter (листинг 14.26). Мы также разрешаем списку ListView выполнять лишь один выбор в заданный момент времени и регистрировать свой OnLongItemClickListener, в результате чего пользователь может установить город в качестве предпочтительного или удалить его.

Метод setCitiesListChangeListener (строки 81–85) обеспечивает для родительского класса Activity настройку слушателя CitiesListChangeListener класса CitiesFragment. Этот слушатель отсылает отчет об изменениях в CitiesFragment объекту WeatherViewerActivity.

Листинг 14.25. Методы onActivityCreated и setCitiesListChangeListener класса CitiesFragment

```
49   // вызывается при создании родительского класса Activity
50   @Override
51   public void onActivityCreated(Bundle savedInstanceStateBundle)
52   {
53      super.onActivityCreated(savedInstanceStateBundle);
54
55      // данный объект Bundle включает информацию о состоянии
56      if (savedInstanceStateBundle != null)
57      {
58         // получает последний выбранный город из объекта Bundle
59         currentCityIndex = savedInstanceStateBundle.getInt(
60            CURRENT_CITY_KEY);
61      }  // конец блока if
62
63      // создание списка ArrayList, используемого для хранения
        // названий городов
64      citiesArrayList = new ArrayList<String>();
65
66      // настройка адаптера ListView класса Fragment
67      setListAdapter(new CitiesArrayAdapter<String>(getActivity(),
68         R.layout.city_list_item, citiesArrayList));
69
70      ListView thisListView = getListView(); // получение ListView
                                               // из Fragment
71      citiesArrayAdapter = (ArrayAdapter<String>)getListAdapter();
72
73      // разрешить выбрать один город в данный момент времени
74      thisListView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
75     thisListView.setBackgroundColor(Color.WHITE); // выбор фонового цвета
76       thisListView.setOnItemLongClickListener(
77          citiesOnItemLongClickListener);
78   }  // конец определения метода onActivityCreated
79
80   // настройка CitiesListChangeListener
81   public void setCitiesListChangeListener(
```

```
      CitiesListChangeListener listener)
   {
      citiesListChangeListener = listener;
   }  // конец определения метода setCitiesChangeListener

```

Вложенный класс CitiesArrayAdapter класса CitiesFragment

Объект CitiesArrayAdapter (листинг 14.26) — это заказной адаптер ArrayAdapter, который отображает название каждого города для элемента списка. Слева от названия предпочтительного города помещается значок «звездочка». Метод getView (строки 101–122), вызываемый для каждого элемента списка ListView класса Fragment, требует новый элемент списка View. Сначала сохраняются результаты вызова метода getView суперкласса, в результате чего обеспечивается повторное использование объекта View (при наличии этого объекта). Название города для данного элемента списка передается методу isPreferredCity (строки 125–136). Если выбран предпочтительный город, отображается значок «звездочка» с помощью метода setCompoundDrawables класса TextView. Если же город не является предпочтительным, используется тот же метод для очистки ранее определенной «звездочки». Метод isPreferredCity возвращает значение true, если данная строка String соответствует названию предпочтительного города. Объект Context родительского объекта Activity обеспечивает доступ к общим настройкам, а затем выполняется сравнение данной строки String с названием предпочтительного города.

Листинг 14.26. Вложенный класс CitiesArrayAdapter класса CitiesFragment

```
 87   // заказной адаптер ArrayAdapter для списка ListView из CitiesFragment
 88   private class CitiesArrayAdapter<T> extends ArrayAdapter<String>
 89   {
 90      private Context context; // объект Context для Activity
                                  // объекта Fragment
 91
 92      // общедоступный конструктор для CitiesArrayAdapter
 93      public CitiesArrayAdapter(Context context, int textViewResourceId,
 94         List<String> objects)
 95      {
 96         super(context, textViewResourceId, objects);
 97         this.context = context;
 98      }  // конец описания конструктора CitiesArrayAdapter
 99
100      // получение элемента ListView для данной позиции
101      @Override
102    public View getView(int position, View convertView, ViewGroup parent)
103      {
104         // получение компонента TextView, генерируемого методом
            // getView из ArrayAdapter
105         TextView listItemTextView = (TextView)
106         super.getView(position, convertView, parent);
107
108         // если этот элемент является предпочтительным городом
109         if (isPreferredCity(listItemTextView.getText().toString()))
```

продолжение ⇨

Листинг 14.24 *(продолжение)*

```
      {
         // отображение «звездочки» справа от первого элемента
         // списка TextView
         listItemTextView.setCompoundDrawablesWithIntrinsicBounds(0, 0,
            android.R.drawable.btn_star_big_on, 0);
      } // конец блока if
      else
      {
         // очистка любых составных графических элементов
         // для элемента списка TextView
         listItemTextView.setCompoundDrawablesWithIntrinsicBounds(0, 0,
            0, 0);
      } // конец блока else
         return listItemTextView;
      } // конец описания метода

      // является ли данный город предпочтительным?
      private boolean isPreferredCity(String cityString)
      {
         // получение SharedPreferences для приложения
         SharedPreferences preferredCitySharedPreferences =
            context.getSharedPreferences(
            WeatherViewerActivity.SHARED_PREFERENCES_NAME,
            Context.MODE_PRIVATE);

      // возвращает true, если данное имя соответствует
      // названию предпочтительного города
     return cityString.equals(preferredCitySharedPreferences.getString(
         WeatherViewerActivity.PREFERRED_CITY_NAME_KEY, null));
      } // конец определения метода isPreferredCity
   }  // конец описания класса CitiesArrayAdapter

```

Реализация интерфейса OnItemLongClickListener

Слушатель citiesOnItemLongClickListener (листинг 14.27) обрабатывает события длительного нажатия элементов ListView класса Fragment. Конструируется диалоговое окно AlertDialog, с помощью которого пользователь может удалить выбранный город или выбрать его в качестве предпочтительного. Метод setPositiveButton класса AlertDialog.Builder применяется для создания параметра Set Preferred. Метод onClick класса OnClickListener для данной кнопки Button (строки 172–177) передает название выбранного города методу onPreferredCityChanged класса CitiesListChangeListener. Метод notifyDataSetChanged класса ArrayAdapter обновляет содержимое компонента ListView. Затем создается объект Button для кнопки Delete, с помощью которой выбранный город удаляется из приложения. С помощью метода onClick (строки 185–233) сначала проверяется, является ли выделенный элемент единственным элементом в списке, использующим метод getCount класса ArrayAdapter. И если это так, он не может быть удален, и соответственно, не может отображаться сообщение Toast. Если же это условие не выполняется, элемент может быть удален с помощью метода remove класса ArrayAdapter. Затем удаляется название

города из набора общих настроек приложения. Если удаленный город являлся предпочтительным, в качестве предпочтительного выбирается первый город в списке. Если же город не был предпочтительным, запрашиваем объект WeatherViewerActivity, отображающий прогноз для предпочтительного города путем передачи его названия методу onSelectedCityChanged класса CitiesListChangeListener.

Листинг 14.27. Реализация интерфейса OnItemLongClickListener

```
// обработка событий, генерируемых в результате длительного
// нажатия элемента ListView
private OnItemLongClickListener citiesOnItemLongClickListener =
   new OnItemLongClickListener()
{
   // вызывается после длительного нажатия элемента ListView
   @Override
   public boolean onItemLongClick(AdapterView<?> listView, View view,
      int arg2, long arg3)
   {
      // получение контекста для данного View
      final Context context = view.getContext();

      // получаем ресурсы для загрузки строк из файла в формате xml
      final Resources resources = context.getResources();

      // получение названия выбранного города
      final String cityNameString =
         ((TextView) view).getText().toString();

      // создание нового диалогового окна AlertDialog
      AlertDialog.Builder builder = new AlertDialog.Builder(context);

      // настройка сообщения AlertDialog
      builder.setMessage(resources.getString(
         R.string.city_dialog_message_prefix) + cityNameString +
         resources.getString(R.string.city_dialog_message_postfix));

      // настройка кнопки "+" в окне AlertDialog
      builder.setPositiveButton(resources.getString(
         R.string.city_dialog_preferred),
         new DialogInterface.OnClickListener()
      {
         @Override
         public void onClick(DialogInterface dialog, int which)
         {
            citiesListChangeListener.onPreferredCityChanged(
               cityNameString);
            citiesArrayAdapter.notifyDataSetChanged();
         }  // конец описания метода onClick
      });// конец описания DialogInterface.OnClickListener
      // настройка нейтральной кнопки в AlertDialog
      builder.setNeutralButton(resources.getString(
```

продолжение ⇗

Листинг 14.27 *(продолжение)*

```
            R.string.city_dialog_delete),
            new DialogInterface.OnClickListener()
         {
             // вызывается после щелчка на кнопке "Delete"
             public void onClick(DialogInterface dialog, int id)
             {
                 // если последний город в списке
                 if (citiesArrayAdapter.getCount() == 1)
                 {
                    // сообщение пользователю о невозможности
                    // удаления последнего города в списке
                    Toast lastCityToast =
                       Toast.makeText(context, resources.getString(
                       R.string.last_city_warning), Toast.LENGTH_LONG);
                    lastCityToast.setGravity(Gravity.CENTER, 0, 0);
                    lastCityToast.show(); // сообщение Toast
                    return; // конец описания метода
                 }  // конец блока if

                 // удаление города из списка
                 citiesArrayAdapter.remove(cityNameString);

                 // получение общих настроек приложения
                 SharedPreferences sharedPreferences =
                    context.getSharedPreferences(
                    WeatherViewerActivity.SHARED_PREFERENCES_NAME,
                    Context.MODE_PRIVATE);

                 // удаление настроек для удаленного города
                 // из  SharedPreferences
                 Editor preferencesEditor = sharedPreferences.edit();
                 preferencesEditor.remove(cityNameString);
                 preferencesEditor.apply();

                 // получение текущего предпочтительного города
                 String preferredCityString =
                    sharedPreferences.getString(
                    WeatherViewerActivity.PREFERRED_CITY_NAME_KEY,
                    resources.getString(R.string.default_zipcode));

                 // если удален предпочтительный город
                 if (cityNameString.equals(preferredCityString))
                 {
                    // установка нового предпочтительного города
                    citiesListChangeListener.onPreferredCityChanged(
                    citiesArrayList.get(0));
                 }  // конец блока if
                 else if (cityNameString.equals(citiesArrayList.get(
                 currentCityIndex)))
                 {
```

```
                  // загрузка прогноза для предпочтительного города
                  citiesListChangeListener.onSelectedCityChanged(
                       preferredCityString);
                  }  // конец блока else if
               }  // конец описания метода onClick
            });// конец описания OnClickListener
            // настройка кнопки отмены в окне AlertDialog
            builder.setNegativeButton(resources.getString(
               R.string.city_dialog_cancel),
               new DialogInterface.OnClickListener()
            {
               // вызывается после щелчка на кнопке "No"
               public void onClick(DialogInterface dialog, int id)
               {
                  dialog.cancel(); // скрытие окна AlertDialog
               }  // конец описания onClick
            });// конец описания OnClickListener

            builder.create().show(); // display the AlertDialog
            return true;
         } // конец описания citiesOnItemLongClickListener
   }; // конец описаниия OnItemLongClickListener

```

Методы onSaveInstanceState, addCity и onListItemClick класса CitiesFragment

Метод onSaveInstanceState (листинг 14.28) сохраняет позицию выбранного элемента CitiesFragment. Метод addCity (строки 263–273) используется классом WeatherViewerActivity для добавления новых городов в компонент ListView. Мы добавляем новую строку String в класс ArrayAdapter, затем сортируем элементы из Adapter в алфавитном порядке. Если параметр select, имеющий тип boolean, равен true, мы передаем название города методу onSelectedCityChanged класса CitiesListChangeListener, в результате чего класс WeatherViewerActivity будет отображать соответствующий прогноз.

Метод onListItemClick (строки 276–283) обрабатывает события щелчков на элементах ListView. Название города выделенных элементов передается методу onSelectedCityChanged класса CitiesListChangeListener, чтобы «проинформировать» WeatherViewerActivity о выборе нового элемента. Затем индекс выбранного элемента списка сохраняется в currentCityIndex.

Листинг 14.28. Методы onSaveInstanceState, addCity и onListItemClick класса CitiesFragment

```
252  // сохранение состояния класса Fragment
253  @Override
254  public void onSaveInstanceState(Bundle outStateBundle)
255  {
256     super.onSaveInstanceState(outStateBundle);
257
```

продолжение ⇗

Листинг 14.28 *(продолжение)*

```
      // сохранение текущего выбранного города в Bundle
      outStateBundle.putInt(CURRENT_CITY_KEY, currentCityIndex);
   }  // конец описания onSaveInstanceState

   // добавление нового города в список
   public void addCity(String cityNameString, boolean select)
   {
      citiesArrayAdapter.add(cityNameString);
      citiesArrayAdapter.sort(String.CASE_INSENSITIVE_ORDER);

      if (select) // если нужно выбрать новый город
      {
         // информирование CitiesListChangeListener
         citiesListChangeListener.onSelectedCityChanged(cityNameString);
      }  // конец блока if
   }  // конец описания метода addCity

   // обработка щелчка на элементе ListView
   @Override
   public void onListItemClick(ListView l, View v, int position, long id)
   {
      // сообщить Activity об обновлении ForecastFragment
      citiesListChangeListener.onSelectedCityChanged(((TextView)v).
         getText().toString());
      currentCityIndex = position; // сохранение текущей
                                   // выбранной позиции
   } // конец описания onListItemClick
} // конец описания класса CitiesFragment
```

14.5.3. Класс AddCityDialogFragment

Класс AddCityDialogFragment (см. листинг 14.29) обеспечивает возможность ввода пользователем почтового индекса для добавления нового города в список избранных городов. Интерфейс DialogFinishedListener (строки 19–23) реализован с помощью класса WeatherViewerActivity (листинг 14.19), в результате чего Activity может получать информацию, вводимую пользователем в AddCityDialogFragment. Интерфейсы обычно используются таким образом для обмена данными между Fragment и родительским классом Activity. Фрагмент DialogFragment включает компонент EditText, с помощью которого пользователь может ввести почтовый индекс, а также флажок CheckBox, после установки которого новый город выбирается как предпочтительный.

Листинг 14.29. Класс AddCityDialogFragment

```
1   // AddCityDialogFragment.java
2   // DialogFragment, предназначенный для ввода пользователем
    // почтового индекса нового города.
3   package com.deitel.weatherviewer;
4
5   import android.app.DialogFragment;
```

```
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;

public class AddCityDialogFragment extends DialogFragment
implements OnClickListener
{
   // выслушивает результаты, полученные из AddCityDialog
   public interface DialogFinishedListener
   {
      // вызывается, если скрывается AddCityDialog
      void onDialogFinished(String zipcodeString, boolean preferred);
   }  // конец описания интерфейса DialogFinishedListener

   EditText addCityEditText; // компонент EditText из DialogFragment
   CheckBox addCityCheckBox; // компонент CheckBox из DialogFragment

   // инициализация нового фрагмента DialogFragment
   @Override
   public void onCreate(Bundle bundle)
   {
      super.onCreate(bundle);

      // обеспечение выхода пользователя с помощью клавиши back
      this.setCancelable(true);
   }  // конец описания метода onCreate

   // "раздувает" разметку DialogFragment
   @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle argumentsBundle)
   {
      // "раздувает" разметку, определенную в файле add_city_dialog.xml
   View rootView = inflater.inflate(R.layout.add_city_dialog, container,
         false);

      // получает  EditText
      addCityEditText = (EditText) rootView.findViewById(
         R.id.add_city_edit_text);

      // получает CheckBox
      addCityCheckBox = (CheckBox) rootView.findViewById(
         R.id.add_city_checkbox);

      if (argumentsBundle != null) // если аргументы Bundle не пусты
```

продолжение ⇨

Листинг 14.29 *(продолжение)*

```
56        {
57           addCityEditText.setText(argumentsBundle.getString(
58              getResources().getString(
59              R.string.add_city_dialog_bundle_key)));
60        }  // конец блока if
61
62        // настройка заголовка DialogFragment
63        getDialog().setTitle(R.string.add_city_dialog_title);
64
65        // инициализация кнопки "+"
66        Button okButton = (Button) rootView.findViewById(
67           R.id.add_city_button);
68        okButton.setOnClickListener(this);
69        return rootView; // возврат корневого View из Fragment
70     }  // конец описания метода onCreateView
71
72     // сохранение текущего состояния DialogFragment
73     @Override
74     public void onSaveInstanceState(Bundle argumentsBundle)
75     {
76        // добавление текста EditText в аргументы Bundle
77        argumentsBundle.putCharSequence(getResources().getString(
78           R.string.add_city_dialog_bundle_key),
79              addCityEditText.getText().toString());
80           super.onSaveInstanceState(argumentsBundle);
81     }  // конец описания метода onSaveInstanceState
82
83     // вызывается после щелчка на кнопке Add City Button
84     @Override
85     public void onClick(View clickedView)
86     {
87        if (clickedView.getId() == R.id.add_city_button)
88        {
89           DialogFinishedListener listener =
90              (DialogFinishedListener) getActivity();
91           listener.onDialogFinished(addCityEditText.getText().toString(),
92               addCityCheckBox.isChecked() );
93           dismiss(); // скрытие DialogFragment
94        }  // конец блока if
95     }  // конец описания метода onClick
96  }  // конец описания класса AddCityDialogFragment
```

Переопределение метода onCreate

Метод onCreate (строки 29–36) переопределяется для вызова метода setCancelable класса DialogFragment. В результате пользователь получает возможность скрыть диалоговое окно DialogFragment, воспользовавшись кнопкой устройства back.

Переопределение метода onCreateView

Разметка класса DialogFragment «раздувается» с помощью метода onCreateView (строки 39–70). В строках 44–53 «раздувается» разметка, определенная в файле add_city_dialog.xml, затем осуществляется выборка компонентов EditText и Checkbox класса DialogFragment. Если устройство поворачивается во время отображения этого диалогового окна, argumentsBundle будет включать любой текст, введенный пользователем в EditText. Благодаря этому возможен поворот DialogFragment без очистки EditText.

Переопределение метода onCreate

Метод onSaveInstanceState (строки 73–81) сохраняет текущее содержимое в EditText, обеспечивая восстановление этого же текста в Fragment в будущем. Чтобы сохранить текст в Bundle, вызывается метод putCharSequence класса argumentBundle.

Переопределение метода onCreate

С помощью метода onClick (строки 84–95) добавляется новый город в список и скрывается окно AddCityDialogFragment после щелчка на кнопке Button из Fragment. Методу onDialogFinished класса DialogFinishedListener передается текст компонента EditText и состояние установки компонента CheckBox. Для удаления этого фрагмента Fragment класса Activity вызывается метод dismiss класса DialogFragment.

14.5.4. Класс ForecastFragment

Абстрактный класс ForecastFragment (листинг 14.30) расширяет класс Fragment и поддерживает абстрактный метод getZipcode, возвращающий строку почтового индекса (ZIP code) типа String. Класс WeatherViewerActivity использует подклассы класса ForecastFragment, которые называются SingleForecastFragment (раздел 14.5.5) и FiveDayForecastFragment (раздел 14.5.8), для отображения текущей погоды и пятидневного прогноза соответственно. В классе WeatherViewerActivity используется метод getZipcode для получения почтового индекса для информации о погоде, отображаемой для каждого типа ForecastFragment.

Листинг 14.30. Класс ForecastFragment

```
// ForecastFragment.java
// Абстрактный класс, определяющий возможности класса Fragment
// по поддержке почтового индекса.
package com.deitel.weatherviewer;

import android.app.Fragment;

public abstract class ForecastFragment extends Fragment
{
   public abstract String getZipcode();
}  // конец определения класса ForecastFragment
```

14.5.5. Класс SingleForecastFragment

Класс SingleForecastFragment является подклассом класса Fragment и предназначен для отображения текущей погоды для данного города.

Операторы package, import и поля экземпляра класса SingleForecastFragment

В листинге 14.31 начинается определение класса SingleForecastFragment и относящихся к нему полей. В строках 25–30 определяются различные константы String, используемые в качестве ключей при сохранении и восстановлении состояния SingleForecastFragment во время изменения ориентации устройства.

Листинг 14.31. Операторы package, import и поля экземпляра класса SingleForecastFragment

```
1  // SingleForecastFragment.java
2  // Отображение текущей погоды для одного города.
3  package com.deitel.weatherviewer;
4
5  import android.content.Context;
6  import android.content.res.Resources;
7  import android.graphics.Bitmap;
8  import android.os.Bundle;
9  import android.view.Gravity;
10 import android.view.LayoutInflater;
11 import android.view.View;
12 import android.view.ViewGroup;
13 import android.widget.ImageView;
14 import android.widget.TextView;
15 import android.widget.Toast;
16
17 import com.deitel.weatherviewer.ReadForecastTask.ForecastListener;
18 import com.deitel.weatherviewer.ReadLocationTask.LocationLoadedListener;
19
20 public class SingleForecastFragment extends ForecastFragment
21 {
22    private String zipcodeString; // почтовый индекс для данного прогноза
23
24    // поиск ключей для сохраненного состояния объекта Fragment
25    private static final String LOCATION_KEY = "location";
26    private static final String TEMPERATURE_KEY = "temperature";
27    private static final String FEELS_LIKE_KEY = "feels_like";
28    private static final String HUMIDITY_KEY = "humidity";
29    private static final String PRECIPITATION_KEY = "chance_precipitation";
30    private static final String IMAGE_KEY = "image";
31
32    // применяется для выборки почтового индекса для сохраненного Bundle
33    private static final String ZIP_CODE_KEY = "id_key";
34
35    private View forecastView; // содержит все виды прогноза
```

```
  private TextView temperatureTextView; // отображение фактической
                                        // температуры
  private TextView feelsLikeTextView;   // отображение температуры
                                        // "по ощущениям"
  private TextView humidityTextView;    // отображение влажности

  private TextView locationTextView;

  // отображение процента вероятности прогноза
  private TextView chanceOfPrecipitationTextView;
  private ImageView conditionImageView; // изображение облачности
  private TextView loadingTextView;
  private Context context;
  private Bitmap conditionBitmap;

```

Перегруженный метод newInstance класса SingleForecastFragment

Статические методы newInstance класса SingleForecastFragment создают и возвращают новый объект Fragment по указанному почтовому индексу. В первой версии этого метода (листинг 14.32, строки 50–64) создается новый объект SingleForecastFragment, затем включается почтовый индекс в новый объект Bundle, который передается методу setArguments класса Fragment. Эта информация в дальнейшем может быть выбрана с помощью переопределенного метода onCreate класса Fragment. Затем метод newInstance берет объект Bundle в качестве аргумента (строки 67–72), считывает почтовый индекс из Bundle и возвращает результат вызова метода newInstance, который использует аргумент типа String.

Листинг 14.32. Перегруженный метод newInstance класса SingleForecastFragment

```
49  // создает новый объект ForecastFragment для данного почтового индекса
50  public static SingleForecastFragment newInstance(String zipcodeString)
51  {
52     // создает новый ForecastFragment
53     SingleForecastFragment newForecastFragment =
54        new SingleForecastFragment();
55
56     Bundle argumentsBundle = new Bundle(); // создание нового Bundle
57
58    // сохранение данной строки в Bundle
59    argumentsBundle.putString(ZIP_CODE_KEY, zipcodeString);
60
61    // настройка аргументов Fragement
62    newForecastFragment.setArguments(argumentsBundle);
63    return newForecastFragment; // return the completed ForecastFragment
64  } // конец определения метода newInstance
65
66  // создание нового ForecastFragment на основе данного Bundle
67 public static SingleForecastFragment newInstance(Bundle argumentsBundle)
```

продолжение ⇨

Листинг 14.32 *(продолжение)*

```
{
   // получение почтового индекса для данного Bundle
   String zipcodeString = argumentsBundle.getString(ZIP_CODE_KEY);
   return newInstance(zipcodeString);//создание нового ForecastFragment
}  // конец определения метода newInstance
```

Методы onCreate, onSaveInstanceState и getZipcode класса SingleForecastFragment

При выполнении метода onCreate (листинг 14.33, строки 75–82) почтовый индекс, определенный строкой типа String, считывается из параметра Bundle и сохраняется в переменной zipcodeString экземпляра класса SingleForecastFragment.

Метод onSaveInstanceState (строки 85–102) сохраняет информацию о прогнозе погоды, которая отображается для текущего объекта Fragment. Благодаря этому не нужно запускать новую задачу AsyncTask при каждом изменении ориентации устройства. Текст, содержащийся в каждом компоненте TextView, добавляется в параметр Bundle с помощью метода putString класса Bundle. Включение объекта Bitmap, представляющего собой иллюстрацию прогноза, осуществляется с помощью метода putParcelable класса Bundle. Метод getZipcode класса ForecastFragment (строки 105–108) возвращает строку String, представляющую почтовый индекс, который связан с данным фрагментом SingleForecastFragment.

Листинг 14.33. Методы onCreate, onSaveInstanceState и getZipcode класса SingleForecastFragment

```
74   // создание объекта Fragment на основе сохраненного состояния Bundle
75   @Override
76   public void onCreate(Bundle argumentsBundle)
77   {
78      super.onCreate(argumentsBundle);
79
80      // получение почтового индекса на основе данного Bundle
81      this.zipcodeString = getArguments().getString(ZIP_CODE_KEY);
82   }  // конец определения метода onCreate
83
84   // сохранение состояния объекта Fragment
85   @Override
86   public void onSaveInstanceState(Bundle savedInstanceStateBundle)
87   {
88      super.onSaveInstanceState(savedInstanceStateBundle);
89
90      // сохранение содержимого View в Bundle
91      savedInstanceStateBundle.putString(LOCATION_KEY,
92         locationTextView.getText().toString());
93      savedInstanceStateBundle.putString(TEMPERATURE_KEY,
94         temperatureTextView.getText().toString());
95      savedInstanceStateBundle.putString(FEELS_LIKE_KEY,
```

```
         feelsLikeTextView.getText().toString());
      savedInstanceStateBundle.putString(HUMIDITY_KEY,
         humidityTextView.getText().toString());
      savedInstanceStateBundle.putString(PRECIPITATION_KEY,
         chanceOfPrecipitationTextView.getText().toString());
      savedInstanceStateBundle.putParcelable(IMAGE_KEY, conditionBitmap);
   }  // конец определения метода onSaveInstanceState

   // общий доступ к почтовому индексу, соответствующему
   // информации о прогнозе для данного объекта Fragment
   public String getZipcode()
   {
      return zipcodeString; // возвращение строки String,
                            // содержащей почтовый индекс
   }  // конец определения метода getZIP code

```

Переопределение метода onCreateView

Метод onCreateView (листинг 14.34) «раздувает» и инициализирует иерархию объектов View класса ForecastFragment. Разметка, определенная в файле forecast_fragment_layout.xml, «раздувается» с помощью данного LayoutInflator. Методу inflate класса LayoutInflator передается значение null в качестве второго аргумента. Этот аргумент обычно определяет объект ViewGroup, с которым связывается только что «раздутый» компонент View. Обратите внимание, что не следует связывать корневой объект View класса Fragment с произвольным объектом ViewGroup с помощью метода onCreateView. Эта операция выполняется позднее автоматически, в жизненном цикле объекта Fragment. С помощью метода findViewById класса View получаем ссылки на каждый из объектов View класса Fragment, а затем возвращаем корневой объект View из разметки.

Листинг 14.34. Переопределние метода onCreateView

```
   // "раздувание" разметки для данного Fragement из xml-файла
   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState)
   {
      // использование данного LayoutInflator для "раздувания"
      // разметки, находящейся в файле forecast_fragment_layout.xml
      View rootView = inflater.inflate(R.layout.forecast_fragment_layout,
         null);

      // получение TextView в иерархии разметок Fragment
      forecastView = rootView.findViewById(R.id.forecast_layout);
      loadingTextView = (TextView) rootView.findViewById(
         R.id.loading_message);
      locationTextView = (TextView) rootView.findViewById(R.id.location);
      temperatureTextView = (TextView) rootView.findViewById(
         R.id.temperature);
      feelsLikeTextView = (TextView) rootView.findViewById(
```

продолжение ☞

Листинг 14.34 *(продолжение)*

```
         R.id.feels_like);
      humidityTextView = (TextView) rootView.findViewById(
         R.id.humidity);
      chanceOfPrecipitationTextView = (TextView) rootView.findViewById(
         R.id.chance_of_precipitation);
      conditionImageView = (ImageView) rootView.findViewById(
         R.id.forecast_image);

      context = rootView.getContext(); // сохранение Context

      return rootView; // возврат к "раздутой" версии View
   }  // конец определения метода onCreateView

```

Переопределение метода onActivityCreated

Метод onActivityCreated (листинг 14.35) вызывается после создания родительских объектов Activity и View для объекта Fragment. При этом проверяется, содержит ли данные параметр Bundle. Если этот параметр данные не содержит, скрываются все компоненты View, отображающие сведения прогноза, и отображается сообщение о загрузке. Затем запускается новая задача ReadLocationTask, с помощью которой вводятся данные для этого объекта Fragment. Если значение параметра Bundle не равно null, выбирается информация, сохраненная в Bundle, с помощью onSaveInstanceState (см. листинг 14.33). Эта информация отображается в компонентах View из объекта Fragment.

Листинг 14.35. Переопределение метода onActivityCreated

```
   // вызывается после создания родительского объекта Activity
   @Override
   public void onActivityCreated(Bundle savedInstanceStateBundle)
   {
      super.onActivityCreated(savedInstanceStateBundle);

      // если сохраненная информация отсутствует
      if (savedInstanceStateBundle == null)
      {
         // скрытие прогноза и отображение сообщения о загрузке
         forecastView.setVisibility(View.GONE);
         loadingTextView.setVisibility(View.VISIBLE);

         // загрузка информации о местоположении в фоновый поток
         new ReadLocationTask(zipcodeString, context,
            new WeatherLocationLoadedListener(zipcodeString)).execute();
      }  // конец блока if
      else
      {
         // отображение информации о сохраненном состоянии Bundle
         // с помощью компонентов View класса Fragment
         conditionImageView.setImageBitmap(
```

```
         (Bitmap) savedInstanceStateBundle.getParcelable(IMAGE_KEY));
      locationTextView.setText(savedInstanceStateBundle.getString(
         LOCATION_KEY));
      temperatureTextView.setText(savedInstanceStateBundle.getString(
         TEMPERATURE_KEY));
      feelsLikeTextView.setText(savedInstanceStateBundle.getString(
         FEELS_LIKE_KEY));
      humidityTextView.setText(savedInstanceStateBundle.getString(
         HUMIDITY_KEY));
      chanceOfPrecipitationTextView.setText(
         savedInstanceStateBundle.getString(PRECIPITATION_KEY));
   }  // конец блока else
}  // конец определения метода onActivityCreated

```

Реализация интерфейса ForecastListener

Интерфейс weatherForecastListener (листинг 14.36) принимает данные от объекта ReadForecastTask (раздел 14.5.7). Сначала проверяется, остается ли связанным этот объект Fragment с объектом WeatherViewerActivity. При этом используется метод isAdded класса Fragment. Если подобная связь отсутствует, пользователь должен отменить выбор объекта Fragment во время выполнения задачи ReadForecastTask (то есть пользователь выходит не выполняя каких-либо действий). Если данные были успешно возвращены, они отображаются с помощью компонентов View класса Fragment.

Листинг 14.36. Реализация интерфейса ForecastListener

```
// получает информацию о погоде из AsyncTask
ForecastListener weatherForecastListener = new ForecastListener()
{
   // отображает сведения о прогнозе погоды
   @Override
   public void onForecastLoaded(Bitmap imageBitmap,
      String temperatureString, String feelsLikeString,
      String humidityString, String precipitationString)
   {
 // если Fragment был отсоединен во время выполнения фонового процесса
      if (!SingleForecastFragment.this.isAdded())
      {
         return; // выйти из метода
   }  // конец блока if
   else if (imageBitmap == null)
   {
      Toast errorToast = Toast.makeText(context,
         context.getResources().getString(
         R.string.null_data_toast), Toast.LENGTH_LONG);
      errorToast.setGravity(Gravity.CENTER, 0, 0);
      errorToast.show(); // show the Toast
      return; // выйти перед обновлением прогноза
   }  // конец блока if

```

продолжение ⇨

Листинг 14.36 *(продолжение)*

```
      Resources resources = SingleForecastFragment.this.getResources();

      // отображение загруженной информации
      conditionImageView.setImageBitmap(imageBitmap);
      conditionBitmap = imageBitmap;
      temperatureTextView.setText(temperatureString + (char)0x00B0 +
         resources.getString(R.string.temperature_unit));
      feelsLikeTextView.setText(feelsLikeString + (char)0x00B0 +
         resources.getString(R.string.temperature_unit));
      humidityTextView.setText(humidityString + (char)0x0025);
      chanceOfPrecipitationTextView.setText(precipitationString +
         (char)0x0025);
      loadingTextView.setVisibility(View.GONE); // скрыть сообщение
                                                // о загрузке
      forecastView.setVisibility(View.VISIBLE); // отображение прогноза
      }  // конец определения метода onForecastLoaded
   }; // конец определения weatherForecastListener

```

Реализация интерфейса LocationLoadedListener

Интерфейс WeatherLocationLoadedListener (листинг 14.37) получает информацию о местоположении из ReadLocationTask (раздел 14.5.6) и отображает строку String, сформированную на основе данных из locationTextView. Затем вызывается на выполнение новая задача ReadForecastTask, которая осуществляет выборку из оставшихся данных прогноза.

Листинг 14.37. Реализация интерфейса LocationLoadedListener

```
   // получает информацию о местоположении от фоновой задачи
   private class WeatherLocationLoadedListener implements
   LocationLoadedListener
   {
      private String zipcodeString; // почтовый индекс для просмотра

      // создание нового интерфейса WeatherLocationLoadedListener
      public WeatherLocationLoadedListener(String zipcodeString)
      {
         this.zipcodeString = zipcodeString;
      }  // конец определения интерфейса WeatherLocationLoadedListener

      // вызывается после загрузки информации о местоположении
      @Override
      public void onLocationLoaded(String cityString, String stateString,
         String countryString)
      {
         if (cityString == null) // если возвращаемые данные отсутствуют
         {
            // отображение сообщения об ошибке
            Toast errorToast = Toast.makeText(
            context, context.getResources().getString(
```

```
            R.string.null_data_toast), Toast.LENGTH_LONG);
            errorToast.setGravity(Gravity.CENTER, 0, 0);
            errorToast.show(); // отображение сообщения Toast
            return; // выход перед обновлением прогноза
         }  // конец блока if
         // отображение информации, возвращенной в TextView
         locationTextView.setText(cityString + " " + stateString + ", " +
            zipcodeString + " " + countryString);
         // загрузка прогноза в фоновый поток
         new ReadForecastTask(zipcodeString, weatherForecastListener,
            locationTextView.getContext()).execute();
         }  // конец определения метода onLocationLoaded
      }  // конец определения класса LocationLoadedListener
   }  // конец определения класса SingleForecastFragment
```

14.5.6. Класс ReadLocationTask

Объект ReadLocationTask осуществляет выборку названия города, штата и страны для данного почтового индекса. Интерфейс LocationLoadedListener описывает возможности слушателя по получению данных о местоположении. Строки, представляющие город, штат и страну, передаются методу слушателя onLocationLoaded после выборки этих данных.

Операторы package, import и поля экземпляра класса ReadLocationTask

В листинге 14.38 начинается определение класса ReadLocationTask, а также определяются переменные экземпляра класса, используемые при считывании местоположения из веб-служб WeatherBug.

Листинг 14.38. Операторы package, import и поля экземпляра класса ReadLocationTask

```
// ReadLocationTask.java
// Считывает информацию о местоположении в фоновом потоке.
package com.deitel.weatherviewer;

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.MalformedURLException;
import java.net.URL;

import android.content.Context;
import android.content.res.Resources;
import android.os.AsyncTask;
import android.util.JsonReader;
import android.util.Log;
import android.view.Gravity;
import android.widget.Toast;

```

продолжение

Листинг 14.38 *(продолжение)*

```
// преобразует почтовый код в название города в фоновом потоке
class ReadLocationTask extends AsyncTask<Object, Object, String>
{
   private static final String TAG = "ReadLocatonTask.java";

private String zipcodeString; // почтовый индекс для местоположения
private Context context; // запуск объекта Context из Activity
private Resources resources; // используется для поиска строк в коде xml

// строки для каждого типа выбираемых данных
private String cityString;
private String stateString;
private String countryString;

// слушатель для выбранной информации
private LocationLoadedListener weatherLocationLoadedListener;

```

Вложенный интерфейс LocationLoadedListener и конструктор ReadLocationTask

Вложенный интерфейс LocationLoadedListener (листинг 14.39, строки 37–41) определяет метод onLocationLoaded, реализуемый еще несколькими классами, которые могут быть извещены о том, что ReadLocationTask принимает ответ от веб-служб WeatherBug. Конструктор ReadLocationTask (строки 44–51) принимает строку почтового индекса типа String, Context из WeatherViewerActivity и LocationLoadedListener. Выполняется сохранение текущего объекта Resources из Context, который может использоваться позднее для загрузки строк из XML-ресурсов приложения.

Листинг 14.39. Вложенный интерфейс LocationLoadedListener и конструктор ReadLocationTask

```
36  // интерфейс приемника информации о местоположении
37  public interface LocationLoadedListener
38  {
39     public void onLocationLoaded(String cityString, String stateString,
40        String countryString);
41  }  // конец определения интерфейса LocationLoadedListener
42
43  // общедоступный конструктор
44  public ReadLocationTask(String zipCodeString, Context context,
45    LocationLoadedListener listener)
46  {
47     this.zipcodeString = zipCodeString;
48     this.context = context;
49     this.resources = context.getResources();
50     this.weatherLocationLoadedListener = listener;
51  }  // конец определения конструктора ReadLocationTask
52
```

Метод doInBackground класса ReadLocationTask Method

С помощью метода doInBackground (листинг 14.40) создается объект InputStreamReader, получающий доступ к веб-службе WeatherBug в местоположении, определенном URL-ссылкой. Этот объект применяется для создания объекта JsonReader, обеспечивающего возможность чтения данных JSON, возвращаемых веб-службой. (Документ JSON можно просмотреть, открыв его непосредственно с помощью weatherServiceURL в окне браузера.) С помощью стандарта JSON (JavaScript Object Notation, Запись объекта JavaScript) обеспечивается простой способ представления объектов JavaScript в виде строк (альтернатива XML, передающего данные между клиентом и сервером). Каждый объект в JSON представлен в виде списка имен свойств и значений, заключенных в фигурные скобки и имеющих следующий формат:

```
{ "ИмяСвойства1" : значение1, "ИмяСвойства2'": значение2 }
```

Массивы представлены JSON с помощью квадратных скобок и имеют следующий формат:

```
[ значение1, значение2, значение3 ]
```

В качестве значений выступают строки, числа, JSON-представление объекта, значения true, false или null. Стандарт JSON обычно используется для обеспечения передачи данных в клиент-серверных взаимодействиях.

Листинг 14.40. Метод doInBackground класса ReadLocationTask

```
// загрузка названия города в фоновый поток
@Override
protected String doInBackground(Object... params)
{
   try
   {
      // создание URL-ссылки на Weatherbug API
      URL url = new URL(resources.getString(
         R.string.location_url_pre_zipcode) + zipcodeString +
         "&api_key=YOUR_API_KEY");

      // создание InputStreamReader на основе URL-ссылки
      Reader forecastReader = new InputStreamReader(
         url.openStream());

      // создание JsonReader на основе Reader
      JsonReader forecastJsonReader = new JsonReader(forecastReader);
      forecastJsonReader.beginObject(); // чтение первого объекта

      // получение следующего названия
      String name = forecastJsonReader.nextName();

      // если имя показывает, что следующий элемент описывает
      // местоположение почтового индекса
      if (name.equals(resources.getString(R.string.location)))
```

продолжение ☞

Листинг 14.40 *(продолжение)*

```
    {
        // начало чтения следующего объекта JSON
        forecastJsonReader.beginObject();

        String nextNameString;

        // если есть дополнительная информация для чтения
        while (forecastJsonReader.hasNext())
       {
          nextNameString = forecastJsonReader.nextName();
          // если имя показывает, что следующий элемент описывает
          // название города, соответствующее почтовому индексу
          if ((nextNameString).equals(
             resources.getString(R.string.city)))
             {
                 // считывание названия города
                 cityString = forecastJsonReader.nextString();
             }   // конец блока if
             else if ((nextNameString).equals(resources.
                getString(R.string.state)))
             {
                stateString = forecastJsonReader.nextString();
             }  // конец блока else if
             else if ((nextNameString).equals(resources.
                getString(R.string.country)))
             {
                countryString = forecastJsonReader.nextString();
             }  // конец блока else if
             else
             {
                forecastJsonReader.skipValue(); // пропуск
                                     // неожиданного значения
             }  // конец блока else
          }  // конец цикла while

          forecastJsonReader.close(); // закрыть JsonReader
       }  // конец блока if
    }  // конец блока try
    catch (MalformedURLException e)
    {
       Log.v(TAG, e.toString()); // вывод исключения в LogCat
    }  // конец блока atch
       catch (IOException e)
       {
          Log.v(TAG, e.toString()); // вывод исключения в LogCat
       }  // конец блока catch

    return null; // если название города не найдено, вернуть null
}  // конец определения метода doInBackground

```

Класс JsonReader включает методы beginObject и beginArray, применяемые для начала чтения объектов и массивов соответственно. В строке 70 метод beginObject класса JsonReader считывает первый объект в документе JSON. С помощью метода nextName класса JsonReader получаем имя из первой пары *имя–значение* объекта (строка 73), затем проверяется соответствие с ожидаемым именем для документа, включающего информацию о местоположении. Если соответствие найдено, происходит перемещение к следующему объекту (строка 80), который описывает информацию о местоположении почтового индекса, а также считывается каждая пара *имя–значение* из объекта с помощью цикла (строки 85–110). Если имя в паре *имя–значение* совпадает с одним из фрагментов данных, используемых для отображения информации о погоде в приложении, соответствующее значение сохраняется в одной из переменных экземпляра класса ReadLocationTask. Класс JsonReader поддерживает методы, применяемые для чтения данных в формате boolean, double, int, long и String. Но поскольку данные отображаются в формате String, используется метод getString класса JsonReader. Все нераспознанные имена пропускаются с помощью метода skipValue класса JsonReader.

ПРИМЕЧАНИЕ

Код, используемый для чтения данных JSON, возвращаемых веб-службами WeatherBug, зависит от структуры возвращаемого документа JSON. Если WeatherBug изменит формат данных JSON в будущем, приложение может сгенерировать исключение.

Метод onPostExecute класса ReadLocationTask

Метод onPostExecute (листинг 14.41) передает результаты потоку GUI для отображения. Если выбираемые данные не null (то есть веб-служба вызывает возвращаемые данные), строки, содержащие сведения о местоположении, передаются сохраненному методу onLocationLoaded класса LocationLoadedListener. Иначе отображается сообщение Toast, информирующее пользователя о неудачной попытке выборки информации о местоположении.

Листинг 14.41. Метод onPostExecute класса ReadLocationTask

```
// выполняется снова в потоке UI после загрузки названия города
protected void onPostExecute(String nameString)
{
   // если найден город, который соответствует почтовому индексу
   if (cityString != null)
   {
      // передача информации обратно LocationLoadedListener
      weatherLocationLoadedListener.onLocationLoaded(cityString,
         stateString, countryString);
   }  // конец блока if
   else
   {
      // отображение сообщения Toast, включающего информацию
      // о местоположении, не найдено
      Toast errorToast = Toast.makeText(context, resources.getString(
```

продолжение ⇨

Листинг 14.41 *(продолжение)*

```
            R.string.invalid_zipcode_error), Toast.LENGTH_LONG);
      errorToast.setGravity(Gravity.CENTER, 0, 0); // центрирование Toast
            errorToast.show(); // показать Toast
      }  // конец блока else
   }  // конец описания метода onPostExecute
} // конец описания класса ReadLocationTask
```

14.5.7. Класс ReadForecastTask

Класс ReadForecastTask получает сведения о текущей погоде для данного почтового индекса.

Операторы package, import и поля экземпляра класса ReadForecastTask

В листинге 14.42 начинается определение класса ReadForecastTask. В строковых переменных экземпляра класса хранится текст, отображаемый в прогнозе погоды. Объект Bitmap хранит иллюстрацию для текущего прогноза погоды. С помощью переменной bitmapSampleSize определяется уменьшение разрешения изображения, определенного с помощью объекта Bitmap.

Интерфейс ForecastListener (строки 37–41) описывает слушателя, который может получать иллюстрации и текст прогноза, хранящиеся в объектах Bitmap и String и представляющие текущую температуру воздуха, температуру «по ощущениям», влажность воздуха и вероятность осадков.

Листинг 14.42. Операторы package, import и поля экземпляра класса ReadForecastTask

```
 1 // ReadForecastTask.java
 2 // Чтение сведений о погоде за пределами главного потока.
 3 package com.deitel.weatherviewer;
 4
 5 import java.io.IOException;
 6 import java.io.InputStreamReader;
 7 import java.io.Reader;
 8 import java.net.MalformedURLException;
 9 import java.net.URL;
10
11 import android.content.Context;
12 import android.content.res.Resources;
13 import android.graphics.Bitmap;
14 import android.graphics.BitmapFactory;
15 import android.os.AsyncTask;
16 import android.util.JsonReader;
17 import android.util.Log;
18
19 class ReadForecastTask extends AsyncTask<Object, Object, String>
20 {
21    private String zipcodeString; // почтовый индекс для
                                    // города с прогнозом погоды
22    private Resources resources;
```

```

   // получает информацию о погоде
   private ForecastListener weatherForecastListener;
   private static final String TAG = "ReadForecastTask.java";

   private String temperatureString; // температура
   private String feelsLikeString;   // температура «по ощущениям»
   private String humidityString;    // влажность
   private String chanceOfPrecipitationString; // вероятность осадков
   private Bitmap iconBitmap; // иллюстрация облачности

   private int bitmapSampleSize = -1;

   // интерфейс получателя сведений о погоде
   public interface ForecastListener
   {
      public void onForecastLoaded(Bitmap image, String temperature,
         String feelsLike, String humidity, String precipitation);
} // конец определения интерфейса ForecastListener

```

Конструктор ReadForecastTask и методы setSampleSize

Конструктор ReadForecastTask (листинг 14.43, строки 44–50) принимает почтовый индекс в формате String, ForecastListener и Context из WeatherViewerActivity.

Метод setSampleSize (строки 53–56) определяет степень уменьшения разрешения при загрузке иллюстрации прогноза, находящейся в Bitmap. Если метод не вызывается, разрешение Bitmap не выполняется. Объект WeatherProvider использует этот метод из-за наличия ограничений на размеры объектов Bitmaps, которые могут передаваться с помощью объекта RemoteViews. Причина появления подобных ограничений заключается в том, что объект RemoteViews обменивается данными процессов с виджетом приложения.

Листинг 14.43. Конструктор ReadForecastTask и методы setSampleSize

```
43 // создается новый объект ReadForecastTask
44 public ReadForecastTask(String zipcodeString,
45    ForecastListener listener, Context context)
46 {
47    this.zipcodeString = zipcodeString;
48    this.weatherForecastListener = listener;
49    this.resources = context.getResources();
50 } // конец определения конструктора ReadForecastTask
51
52 // настройка размера выборки для Bitmap прогноза
53 public void setSampleSize(int sampleSize)
54 {
55    this.bitmapSampleSize = sampleSize;
56 } // конец определения метода setSampleSize
57
```

Методы doInBackground и onPostExecute класса ReadForecastTask

Метод doInBackground method (листинг 14.44, строки 59–101) получает и анализирует документ JSON из веб-службы WeatherBug, представляющий текущую погоду, в фоновом потоке. Создается URL-ссылка, указывающая на веб-службу, которая затем используется для конструирования JsonReader. Методы beginObject и nextName класса JsonReader используются для чтения первого имени первого объекта в документе (строки 75 и 78), если имя соответствует строке String, указанной в ресурсе R.string.hourly_forecast типа String. Затем JsonReader передается методу readForecast для выполнения анализа прогноза. Метод onPostExecute (строки 104–110) возвращает выбранные строки Strings методу onForecastLoaded класса ForecastLoadedListener для отображения.

Листинг 14.44. Методы doInBackground и onPostExecute класса ReadForecastTask

```
58  // загрузка прогноза в фоновом потоке
59  protected String doInBackground(Object... args)
60  {
61     try
62     {
63        // url-ссылка для службы JSON WeatherBug
64        URL webServiceURL = new URL(resources.getString(
65           R.string.pre_zipcode_url) + zipcodeString + "&ht=t&ht=i&"
66           + "ht=cp&ht=fl&ht=h&api_key=YOUR_API_KEY");
67
68        // создание потока Reader на основе url-ссылки WeatherBug
69        Reader forecastReader = new InputStreamReader(
70           webServiceURL.openStream());
71
72        // создание JsonReader из Reader
73        JsonReader forecastJsonReader = new JsonReader(forecastReader);
74
75        forecastJsonReader.beginObject(); // чтение первого объекта
76
77        // получение следующего имени
78        String name = forecastJsonReader.nextName();
79
80        // если это ожидаемое имя данных, относящихся
          // к почасовому прогнозу погоды
81        if (name.equals(resources.getString(R.string.hourly_forecast)))
82        {
83           readForecast(forecastJsonReader); // чтение прогноза
84        }  // конец блока if
85
86        forecastJsonReader.close(); // закрыть JsonReader
87     }  // конец блока try
88     catch (MalformedURLException e)
89     {
90        Log.v(TAG, e.toString());
91     }  // конец блока catch
92        catch (IOException e)
```

```
   {
      Log.v(TAG, e.toString());
   }  // конец блока catch
   catch (IllegalStateException e)
   {
      Log.v(TAG, e.toString() + zipcodeString);
   }  // конец блока catch
      return null;
}  // конец определения метода doInBackground

// обновление UI снова в основном потоке
protected void onPostExecute(String forecastString)
{
   // передача информации ForecastListener
   weatherForecastListener.onForecastLoaded(iconBitmap,
      temperatureString, feelsLikeString, humidityString,
      chanceOfPrecipitationString);
}  // конец определения метода onPostExecute

```

Метод getIconBitmap класса ReadForecastTask

Статический метод getIconBitmap (листинг 14.45) преобразует строку погодных условий (String) в объект Bitmap. Документ JSON WeatherBug определяет относительный путь к изображению прогноза, находящемуся на веб-сайте WeatherBug. Создается URL-ссылка, указывающая на местонахождение изображения. Изображение загружается с сервера WeatherBug с помощью статического метода decodeStream класса BitmapFactory.

Листинг 14.45. Метод getIconBitmap класса ReadForecastTask

```
// получение иллюстрации Bitmap текущего состояния (облачности) неба
public static Bitmap getIconBitmap(String conditionString,
   Resources resources, int bitmapSampleSize)
{
   Bitmap iconBitmap = null; // создание объекта Bitmap
   try
   {
      // создание URL-ссылки на изображение, находящееся
      // на сайте WeatherBug
      URL weatherURL = new URL(resources.getString(
         R.string.pre_condition_url) + conditionString +
         resources.getString(R.string.post_condition_url));

      BitmapFactory.Options options = new BitmapFactory.Options();
      if (bitmapSampleSize != -1)
      {
         options.inSampleSize = bitmapSampleSize;
      }  // конец блока if

      // сохранение изображения как Bitmap
      iconBitmap = BitmapFactory.decodeStream(weatherURL.
```

продолжение ⇨

Листинг 14.45 *(продолжение)*

```
            openStream(), null, options);
      } // конец блока try
      catch (MalformedURLException e)
      {
         Log.e(TAG, e.toString());
      } // конец блока catch
      catch (IOException e)
      {
         Log.e(TAG, e.toString());
      } // конец блока catch

      return iconBitmap; // возврат изображения
   } // конец определения метода getIconBitmap

```

Метод readForecast класса ReadForecastTask

Метод readForecast (листинг 14.46) анализирует прогноз текущей погоды, используя параметр JsonReader. Методы beginArray и beginObject класса JsonReader (строки 151–152) используются для начала чтения первого объекта следующего массива в документе JSON. Затем выполняется циклический обход каждого имени в объекте и сравнение с ожидаемыми названиями отображаемой информации. Метод skipValue класса JsonReader используется для пропуска информации, которая не требуется.

Листинг 14.46. Метод readForecast класса ReadForecastTask

```
146   // чтение информации прогноза с помощью данного JsonReader
147   private String readForecast(JsonReader reader)
148   {
149      try
150      {
151         reader.beginArray();  // начало чтения следующего массива
152         reader.beginObject(); // начало чтения следующего объекта
153
154         // пока есть следующий элемент в текущем объекте
155         while (reader.hasNext())
156         {
157            String name = reader.nextName(); // чтение следующего имени
158
159            // если этот элемент temperature
160            if (name.equals(resources.getString(R.string.temperature)))
161            {
162               // чтение элемента temperature (температура)
163               temperatureString = reader.nextString();
164            } // конец блока if
165               // если этот элемент "feels-like" temperature
                  // (температура «по ощущениям»)
166          else if (name.equals(resources.getString(R.string.feels_like)))
167            {
```

```
               // чтение элемента "feels-like" temperature
               // (температура «по ощущениям»)
               feelsLikeString = reader.nextString();
            }  // конец блока else if
               // если этот элемент humidity (влажность)
         else if (name.equals(resources.getString(R.string.humidity)))
            {
               humidityString = reader.nextString(); // чтение humidity
            }  // конец блока else if
               // если следующий элемент chance of precipitation
               // (вероятность осадков)
         else if (name.equals(resources.getString(
            R.string.chance_of_precipitation)))
            {
               // чтение элемента chance of precipitation
               chanceOfPrecipitationString = reader.nextString();
            }  // конец блока else if
               // если следующий элемент icon name (имя значка)
            else if (name.equals(resources.getString(R.string.icon)))
            {
                // чтение элемента icon name
            iconBitmap = getIconBitmap(reader.nextString(), resources,
                   bitmapSampleSize);
            }  // конец блока else if
            else // непредвиденный элемент
            {
               reader.skipValue(); // пропуск следующего элемента
            }  // конец блока else
         }  // конец блока while
      }  // конец блока try
      catch (IOException e)
      {
         Log.e(TAG, e.toString());
      }  // конец блока catch
         return null;
      }  // конец описания метода readForecast
   }  // конец описания ReadForecastTask
```

14.5.8. Класс FiveDayForecastFragment

Класс FiveDayForecastFragment отображает пятидневный прогноз для одного города.

Операторы package, import и поля экземпляра класса FiveDayForecastFragment

В листинге 14.47 начинается определение класса FiveDayForecastFragment, а также определяются поля, используемые в классе.

Листинг 14.47. Операторы package, import и поля экземпляра класса FiveDayForecastFragment

```
// FiveDayForecastFragment.java
// Отображает пятидневный прогноз для одного города.
package com.deitel.weatherviewer;

import android.content.Context;
import android.content.res.Configuration;
import android.os.Bundle;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.deitel.weatherviewer.ReadFiveDayForecastTask.
   FiveDayForecastLoadedListener;
import com.deitel.weatherviewer.ReadLocationTask.LocationLoadedListener;

public class FiveDayForecastFragment extends ForecastFragment
{
   // используется для выборки почтового индекса для сохраненного Bundle
   private static final String ZIP_CODE_KEY = "id_key";
   private static final int NUMBER_DAILY_FORECASTS = 5;

   private String zipcodeString; // почтовый индекс для этого прогноза
   private View[] dailyForecastViews = new View[NUMBER_DAILY_FORECASTS];

   private TextView locationTextView;

```

Перегруженные методы newInstance класса FiveDayForecastFragment

Как и в случае с классом SingleForecastFragment, поддерживается перегруженный метод newInstance (листинг 14.48) для создания новых FiveDayForecastFragments. Первый метод (строки 32–46) использует почтовый индекс типа String. Другой метод (строки 49–55) использует параметр Bundle, содержащий строку почтового индекса типа String, извлекает почтовый индекс и передает его первому методу. В строках 38 и 41 создается и конфигурируется объект Bundle, включающий строку почтового индекса типа String, который затем передается методу setArguments класса Fragment, который может использоваться методом onCreate (листинг 14.49).

Листинг 14.48. Перегруженные методы newInstance класса FiveDayForecastFragment

```
31   // создает новый FiveDayForecastFragment для данного почтового индекса
32   public static FiveDayForecastFragment newInstance(String zipcodeString)
33   {
```

```
      // создается новый ForecastFragment
      FiveDayForecastFragment newFiveDayForecastFragment =
         new FiveDayForecastFragment();

      Bundle argumentsBundle = new Bundle(); // создание нового
                                             // объекта Bundle

      // сохранение данной строки String в Bundle
      argumentsBundle.putString(ZIP_CODE_KEY, zipcodeString);

      // настройка аргументов класса Fragement
      newFiveDayForecastFragment.setArguments(argumentsBundle);
      return newFiveDayForecastFragment;// возврат к завершенному Fragment
   } // конец определения метода newInstance

   // создание объекта FiveDayForecastFragment с помощью данного Bundle
   public static FiveDayForecastFragment newInstance(
      Bundle argumentsBundle)
      {
         // получение почтового индекса из данного Bundle
         String zipcodeString = argumentsBundle.getString(ZIP_CODE_KEY);
        return newInstance(zipcodeString); // создание нового Fragment
   } // конец определения метода newInstance

```

Методы onCreate и getZipCode класса FiveDayForecastFragment

Почтовый индекс считывается методом onCreate класса Fragment (см. листинг 14.49, строки 58–65). Метод getArguments класса Fragment осуществляет выборку Bundle, затем метод getString класса Bundle получает доступ к почтовому индексу типа String. Метод getZipcode (строки 68–71) вызывается классом WeatherViewerActivity для получения почтового индекса из FiveDayForecastFragment.

Листинг 14.49. Методы onCreate и getZipCode класса FiveDayForecastFragment

```
57   // создание объекта Fragment на Bundle для сохраненного состояния
58   @Override
59   public void onCreate(Bundle argumentsBundle)
60   {
61      super.onCreate(argumentsBundle);
62
63      // получение почтового индекса из текущего Bundle
64      this.zipcodeString = getArguments().getString(ZIP_CODE_KEY);
65   } // конец описания метода onCreate
66
67   // общий доступ к почтовому индексу для информации
     // о прогнозе для данного объекта Fragment
68   public String getZipcode()
69   {
70      return zipcodeString; // возврат почтового индекса типа String
71   } // конец описания метода getZipcode
72
```

Метод onCreateView класса FiveDayForecastFragment

Разметка объекта Fragment создана с помощью метода onCreateView (листинг 14.50). «Раздувается» разметка, определенная в файле five_day_forecast.xml с помощью данного метода LayoutInflator, в качестве второго аргумента которого передается null. Здесь же проверяется ориентация устройства для определения макета, используемого для каждого компонента View, включающего ежедневный прогноз. Затем выполняется «раздувание» пяти выбранных разметок и добавление каждого компонента View в контейнер LinearLayout. После этого вызывается метод ReadLocationTask, выполняющий выборку информации о местоположении для соответствующего города из данного объекта Fragment.

Листинг 14.50. Метод onCreateView класса FiveDayForecastFragment

```
// «раздувает» разметку объекта Fragement из xml-файла
@Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState)
   {
   // «раздувает» макет для пятидневного прогноза
   View rootView = inflater.inflate(R.layout.five_day_forecast_layout,
      null);
   // получение TextView, отображающего информацию о местоположении
   locationTextView = (TextView) rootView.findViewById(R.id.location);

   // получение ViewGroup, включающего разметки
   // для ежедневного прогноза
   LinearLayout containerLinearLayout =
      (LinearLayout) rootView.findViewById(R.id.containerLinearLayout);

   int id; // идентификатор int для разметки ежедневного прогноза

   // если выбрана альбомная ориентация
   if (container.getContext().getResources().getConfiguration().
      orientation == Configuration.ORIENTATION_LANDSCAPE)
   {
      id = R.layout.single_forecast_layout_landscape;
   }  // конец блока if
   else // портретная ориентация
   {
      id = R.layout.single_forecast_layout_portrait;
      containerLinearLayout.setOrientation(LinearLayout.VERTICAL);
   }  // конец блока else

   // загрузка пятидневных прогнозов погоды
   View forecastView;
      for (int i = 0; i < NUMBER_DAILY_FORECASTS; i++)
      {
         forecastView = inflater.inflate(id, null); // «раздувание»
                                                    // нового View
```

```

            // добавление нового View в контейнер LinearLayout
            containerLinearLayout.addView(forecastView);
            dailyForecastViews[i] = forecastView;
         }  // конец цикла for

         // загрузка сведений о местоположении в фоновый поток
         new ReadLocationTask(zipcodeString, rootView.getContext(),
            new WeatherLocationLoadedListener(zipcodeString,
            rootView.getContext())).execute();

         return rootView;
     }  // конец определения метода onCreateView

```

Реализация интерфейса LocationLoadedListener

Метод WeatherLocationLoadedListener класса FiveDayForecastFragment (листинг 14.51) подобен другим методам LocationLoadedListener в этом приложении. Он получает данные из ReadLocationTask и отображает форматированную строку (String), содержащую информацию о местоположении, с помощью компонента TextView.

Листинг 14.51. Реализация интерфейса LocationLoadedListener

```
// получает сведения о местоположении от фоновой задачи
private class WeatherLocationLoadedListener implements
   LocationLoadedListener
{
   private String zipcodeString; // просматриваемый почтовый индекс
   private Context context;

   // создание нового слушателя WeatherLocationLoadedListener
   public WeatherLocationLoadedListener(String zipcodeString,
      Context context)
   {
      this.zipcodeString = zipcodeString;
      this.context = context;
   }  // конец определения WeatherLocationLoadedListener

   // вызывается, если загружена информация о местоположении
   @Override
   public void onLocationLoaded(String cityString, String stateString,
         String countryString)
      {
         if (cityString == null) // если нет возвращаемых данных
         {
             // отображение сообщения об ошибке
             Toast errorToast = Toast.makeText(context,
          context.getResources().getString(R.string.null_data_toast),
                Toast.LENGTH_LONG);
             errorToast.setGravity(Gravity.CENTER, 0, 0);
```

продолжение ⇨

Листинг 14.51 *(продолжение)*

```
                errorToast.show(); // показать Toast
                return; // выход перед обновлением прогноза
            }  // конец блока if

            // отображение информации, возвращаемой TextView
         locationTextView.setText(cityString + " " + stateString + ", " +
               zipcodeString + " " + countryString);

            // загрузка прогноза в фоновом потоке
            new ReadFiveDayForecastTask(
               weatherForecastListener,
               locationTextView.getContext()).execute();
       } // конец описания метода onLocationLoaded
   }  // конец описания класса WeatherLocationLoadedListener

```

Реализация интерфейса FiveDayForecastLoadedListener

Интерфейс FiveDayForecastLoadedListener (листинг 14.52) получает массив, состоящий из пяти объектов DailyForecast, с помощью метода onForecastLoaded. Информация в DailyForecasts отображается путем ее передачи методу loadForecastIntoView (листинг 14.53).

Листинг 14.52. Реализация интерфейса FiveDayForecastLoadedListener

```
   // получает информацию о погоде от AsyncTask
   FiveDayForecastLoadedListener weatherForecastListener =
      new FiveDayForecastLoadedListener()
   {
      // если завершена фоновая задача по поиску
      // сведений о местоположении
      @Override
      public void onForecastLoaded(DailyForecast[] forecasts)
     {
        // отображение пятидневных прогнозов
        for (int i = 0; i < NUMBER_DAILY_FORECASTS; i++)
        {
           // отображение информации о прогнозе
           loadForecastIntoView(dailyForecastViews[i], forecasts[i]);
        }  // конец цикла for
     }  // конец описания метода onForecastLoaded
  }; // конец описания FiveDayForecastLoadedListener

```

Метод loadForecastIntoView класса FiveDayForecastFragment

Метод loadForecastIntoView (листинг 14.53) отображает сведения в текущем DailyForecast, используя компонент View. Если этот объект Fragment остается прикрепленным к WeatherViewerActivity и DailyForecast не пуст, получаем ссылки на каждый дочерний

компонент View в данном объекте ViewGroup. Дочерние компоненты View применяются для отображения каждого элемента данных в DailyForecast.

Листинг 14.53. Метод loadForecastIntoView класса FiveDayForecastFragment

```
// отображение данной информации прогноза в данном View
private void loadForecastIntoView(View view,
   DailyForecast dailyForecast)
{
   // если Fragment отсоединен во время выполнения фонового процесса
   if (!FiveDayForecastFragment.this.isAdded())
   {
      return; // оставить метод
   }  // конец блока if
      // если отсутствуют возвращенные данные
   else if (dailyForecast == null ||
   dailyForecast.getIconBitmap() == null)
   {
      // отображение сообщения об ошибке
      Toast errorToast = Toast.makeText(view.getContext(),
         view.getContext().getResources().getString(
         R.string.null_data_toast), Toast.LENGTH_LONG);
       errorToast.setGravity(Gravity.CENTER, 0, 0);
       errorToast.show(); // отобразить Toast
       return; // выход перед обновлением прогноза
   }  // конец блока else if

      // получение всех дочерних View
      ImageView forecastImageView = (ImageView) view.findViewById(
         R.id.daily_forecast_bitmap);
      TextView dayOfWeekTextView = (TextView) view.findViewById(
         R.id.day_of_week);
      TextView descriptionTextView = (TextView) view.findViewById(
         R.id.daily_forecast_description);
      TextView highTemperatureTextView = (TextView) view.findViewById(
         R.id.high_temperature);
      TextView lowTemperatureTextView = (TextView) view.findViewById(
          R.id.low_temperature);

      // отображение сведений о прогнозе погоды в выбранных View
      forecastImageView.setImageBitmap(dailyForecast.getIconBitmap());
      dayOfWeekTextView.setText(dailyForecast.getDay());
      descriptionTextView.setText(dailyForecast.getDescription());
  highTemperatureTextView.setText(dailyForecast.getHighTemperature());
   lowTemperatureTextView.setText(dailyForecast.getLowTemperature());
   }  // конец определения метода loadForecastIntoView
}  // конец определения класса FiveDayForecastFragment
```

14.5.9. Класс ReadFiveDayForecastTask

Класс ReadFiveDayForecastTask относится к категории AsyncTask, он использует JsonReader для загрузки пятидневных прогнозов из веб-службы WeatherBug.

Операторы package, import из ReadFiveDayForecastTask, поля и вложенный интерфейс FiveDayForecastLoadedListener

В листинге 14.54 начинается определение класса ReadFiveDayForecastTask и определение полей, используемых в классе. Интерфейс FiveDayForecastLoadedListener (строки 30–33) описывает слушателя, который может принимать пятидневные DailyForecasts при возвращении фоновой задачей данных в поток GUI для их отображения.

Листинг 14.54. Класс ReadFiveDayForecast

```
1  // ReadFiveDayForecastTask.java
2  // Чтение следующего 5-дневного прогноза в фоновом процессе.
3  package com.deitel.weatherviewer;
4
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.Reader;
8  import java.net.MalformedURLException;
9  import java.net.URL;
10
11 import android.content.Context;
12 import android.content.res.Resources;
13 import android.content.res.Resources.NotFoundException;
14 import android.graphics.Bitmap;
15 import android.os.AsyncTask;
16 import android.util.JsonReader;
17 import android.util.Log;
18
19 class ReadFiveDayForecastTask extends AsyncTask<Object, Object, String>
20 {
21    private static final String TAG = "ReadFiveDayForecastTask";
22
23    private String zipcodeString;
24    private FiveDayForecastLoadedListener weatherFiveDayForecastListener;
25    private Resources resources;
26    private DailyForecast[] forecasts;
27    private static final int NUMBER_OF_DAYS = 5;
28
29    // интерфейс приемника сведений о погоде
30    public interface FiveDayForecastLoadedListener
31    {
32       public void onForecastLoaded(DailyForecast[] forecasts);
33  }  // конец определения интерфейса FiveDayForecastLoadedListener
34
```

Конструктор ReadFiveDayForecastTask

Конструктор ReadFiveDayForecastTask (листинг 14.55) принимает zipcodeString для выбранного города, FiveDayForecastLoadedListener и объект Context из WeatherViewerActivity. Инициализируется массив, предназначенный для хранения пяти DailyForecasts.

Листинг 14.55. Конструктор ReadFiveDayForecast

```
// создает новый объект ReadForecastTask
public ReadFiveDayForecastTask(String zipcodeString,
FiveDayForecastLoadedListener listener, Context context)
{
   this.zipcodeString = zipcodeString;
   this.weatherFiveDayForecastListener = listener;
   this.resources = context.getResources();
   this.forecasts = new DailyForecast[NUMBER_OF_DAYS];
}  // конец определения конструктора ReadFiveDayForecastTask

```

Метод doInBackground класса ReadFiveDayForecastTask

Метод doInBackground (листинг 14.56) вызывает на выполнение веб-службу в отдельном потоке. Создается InputStreamReader, получающий доступ к веб-службе WeatherBug в локации, описанной с помощью webServiceURL. После получения доступа к первому объекту в документе JSON (строка 62) считывается следующее имя, которое должно описывать список прогнозов. Затем начинаем считывать следующий массив (строка 70) и вызываем метод skipValue класса forecastJsonRead для попуска следующего объекта. В результате пропускаются все значения первого объекта, который описывает текущую погоду. После чего вызывается readDailyForecast для следующих пяти объектов, которые содержат следующие пять ежедневных- прогнозов.

Листинг 14.56. Метод doInBackground класса ReadFiveDayForecastTask

```
45   @Override
46   protected String doInBackground(Object... params)
47   {
48      // url-ссылка для службы JSON WeatherBug
49      try
50      {
51         URL webServiceURL = new URL("http://i.wxbug.net/REST/Direct/" +
52            "GetForecast.ashx?zip="+ zipcodeString + "&ht=t&ht=i&"
53            + "nf=7&ht=cp&ht=fl&ht=h&api_key=YOUR_API_KEY");
54
55         // создание Reader для потока на основе url-ссылки WeatherBug
56         Reader forecastReader = new InputStreamReader(
57            webServiceURL.openStream());
58
59         // создание JsonReader на основе Reader
60         JsonReader forecastJsonReader = new JsonReader(forecastReader);
61
```

продолжение ☞

Листинг 14.56 *(продолжение)*

```
         forecastJsonReader.beginObject(); // чтение следующего Object

         // получение следующего имени
         String name = forecastJsonReader.nextName();

         // если его имя ожидается в информации ежедневного прогноза
         if (name.equals(resources.getString(R.string.forecast_list)))
         {
            forecastJsonReader.beginArray(); // начало чтения
                                             // первого массива
            forecastJsonReader.skipValue(); // пропуск сегодняшнего
                                            // прогноза

            // чтение следующих пяти ежедневных прогнозов
            for (int i = 0; i < NUMBER_OF_DAYS; i++)
            {
               // начало чтения следующего объекта
               forecastJsonReader.beginObject();

               // если есть больше данных
               if (forecastJsonReader.hasNext())
               {
                   // чтение следующего прогноза
                   forecasts[i] = readDailyForecast(forecastJsonReader);
               }  // конец блока if
            }  // конец цикла for
         }  // конец блока if

         forecastJsonReader.close(); // закрытие JsonReader

      }  // конец блока try
      catch (MalformedURLException e)
      {
         Log.v(TAG, e.toString());
      }  // конец блока catch
      catch (NotFoundException e)
      {
         Log.v(TAG, e.toString());
      } // конец блока catch
      catch (IOException e)
      {
         Log.v(TAG, e.toString());
      }  // конец блока catch
      return null;
   }  // конец описания метода doInBackground

```

Методы readDailyForecast и onPostExecute класса ReadFiveDayForecastTask

С помощью метода readDailyForecast (листинг 14.57, строки 107–161) читается и обрабатывается каждый объект JSON прогноза. Для хранения всех данных, относящихся к прогнозу, создается новый массив String, включающий четыре элемента и объект Bitmap. С помощью метода hasNext класса forecastReader проверяется наличие элементов объекта, которые еще не были прочитаны. При наличии подобных элементов читается следующее имя, а затем проверяется, соответствует ли оно одному из фрагментов отображаемых данных. Если подобное соответствие найдено, с помощью метода nextString класса JsonReader считывается соответствующее значение. Для получения объекта Bitmap с веб-сайта WeatherBug методу getIconBitmap передается строка значка. С помощью метода skipValue класса JsonReader пропускаются значения нераспознанных имен. Объекты DailyForecast инкапсулируют информацию о погоде для каждого дня.

Метод onPostExecute (строки 164–167) возвращает результаты потоку GUI для дальнейшего отображения. С помощью метода onForecastLoaded класса FiveDayForecastListener массив DailyForecasts передается обратно объекту FiveDayForecastFragment.

Листинг 14.57. Методы readDailyForecast и onPostExecute класса ReadFiveDayForecastTask

```
106   // чтение ежедневного прогноза
107   private DailyForecast readDailyForecast(JsonReader forecastJsonReader)
108   {
109      // создание массива для хранения информации о прогнозах
110      String[] dailyForecast = new String[4];
111      Bitmap iconBitmap = null; // хранение иллюстрации прогноза
112
113      try
114      {
115         // пока есть следующий элемент в текущем объекте
116         while (forecastJsonReader.hasNext())
117         {
118            String name = forecastJsonReader.nextName(); // чтение
                                                           // следующего имени
119
120            if (name.equals(resources.getString(R.string.day_of_week)))
121            {
122               dailyForecast[DailyForecast.DAY_INDEX] =
123                  forecastJsonReader.nextString();
124            }  // конец блока if
125            else if (name.equals(resources.getString(
126               R.string.day_prediction)))
127               {
128                  dailyForecast[DailyForecast.PREDICTION_INDEX] =
129                     forecastJsonReader.nextString();
130               }  // конец блока end else if
131               else if (name.equals(resources.getString(R.string.high)))
```

продолжение ➪

Листинг 14.57 *(продолжение)*

```
            {
               dailyForecast[DailyForecast.HIGH_TEMP_INDEX] =
                  forecastJsonReader.nextString();
            }  // конец блока end else if
            else if (name.equals(resources.getString(R.string.low)))
            {
               dailyForecast[DailyForecast.LOW_TEMP_INDEX] =
                  forecastJsonReader.nextString();
            }  // конец блока else if
            // если следующий элемент — имя значка
         else if (name.equals(resources.getString(R.string.day_icon)))
            {
               // чтение имени значка
               iconBitmap = ReadForecastTask.getIconBitmap(
                  forecastJsonReader.nextString(), resources, 0);
            }  // конец блока else if
            else // если есть непредвиденный элемент
            {
               forecastJsonReader.skipValue(); //пропуск
                                                  // следующего элемента
            }  // конец блока else
         }  // конец цикла while
         forecastJsonReader.endObject()
      } // конец блока try
      catch (IOException e)
      {
         Log.e(TAG, e.toString());
      }  // конец блока catch

      return new DailyForecast(dailyForecast, iconBitmap);
   }  // конец описания метода readDailyForecast

   // обновление UI в основном потоке
   protected void onPostExecute(String forecastString)
   {
      weatherFiveDayForecastListener.onForecastLoaded(forecasts);
   }  // конец описания метода onPostExecute
} // конец описания класса ReadFiveDayForecastTask
```

14.5.10. Класс DailyForecast

Класс DailyForecast (листинг 14.58) инкапсулирует информацию, относящуюся к однодневному прогнозу погоды. Класс определяет четыре общедоступных константы-индекса, используемых для получения информации из массива типа String, в котором хранятся данные о погоде. Объект iconBitmap типа Bitmap хранит изображение, иллюстрирующее прогноз.

Конструктор DailyForecast использует массив String, предполагая, что он отсортирован в корректном порядке, в результате чего константы-индексы соответствуют корректным

базовым данным. Для каждого фрагмента данных, хранящегося в DailyForecast, также поддерживаются общедоступные методы accessor.

Листинг 14.58. Класс DailyForecast

```
// DailyForecast.java
// Представляет однодневный прогноз.
package com.deitel.weatherviewer;

import android.graphics.Bitmap;

public class DailyForecast
{
   // индексы для всей информации о прогнозах
   public static final int DAY_INDEX = 0;
   public static final int PREDICTION_INDEX = 1;
   public static final int HIGH_TEMP_INDEX = 2;
   public static final int LOW_TEMP_INDEX = 3;

   final private String[] forecast;  // массив информации
                                     // для всех прогнозов
   final private Bitmap iconBitmap;  // изображение, представляющее
                                     // прогноз

   // создание нового DailyForecast
   public DailyForecast(String[] forecast, Bitmap iconBitmap)
   {
      this.forecast = forecast;
      this.iconBitmap = iconBitmap;
   }  // конец определения DailyForecast

   // получение иллюстрации для прогноза
   public Bitmap getIconBitmap()
   {
      return iconBitmap;
   }  // конец определения метода getIconBitmap

   // получение дня недели прогноза
   public String getDay()
   {
      return forecast[DAY_INDEX];
   }  // конец определения метода getDay

   // получение краткого описания прогноза
   public String getDescription()
   {
      return forecast[PREDICTION_INDEX];
   }  // конец описания метода getDescription

   // возврат максимальной температуры прогноза
   public String getHighTemperature()
```

продолжение ⇨

Листинг 14.58 *(продолжение)*

```
  {
     return forecast[HIGH_TEMP_INDEX];
  }  // конец определения метода getHighTemperature

     // возврат минимальной температуры прогноза
     public String getLowTemperature()
     {
        return forecast[LOW_TEMP_INDEX];
     }  // конец описания метода getLowTemperature
}  // конец описания класса DailyForecast
```

14.5.11. Класс WeatherProvider

Класс WeatherProvider расширяет класс AppWidgetProvider, обновляя виджет приложения Weather Viewer. Объекты AppWidgetProviders представляют собой специальные объекты BroadcastReceivers, которые прослушивают все широковещательные трансляции, связанные с виджетом этого приложения.

Операторы package, import и константа класса WeatherProvider

В листинге 14.59 начинается определение класса ReadFiveDayForecastTask, а также полей, используемых в этом классе. Константа BITMAP_SAMPLE_SIZE предназначена для уменьшения Bitmap до размера, который может быть использован вместе с RemoteViews — иерархей компонентов View, которые могут быть отображены в другом процессе. В Android ограничивается объем данных, которые могут передаваться между процессами.

Листинг 14.59. Операторы package, import и константа класса WeatherProvider

```
// WeatherProvider.java
// Обновления виджета приложения Weather
package com.deitel.weatherviewer;

import android.app.IntentService;
import android.app.PendingIntent;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.res.Resources;
import android.graphics.Bitmap;
import android.widget.RemoteViews;
import android.widget.Toast;

import com.deitel.weatherviewer.ReadForecastTask.ForecastListener;
import com.deitel.weatherviewer.ReadLocationTask.LocationLoadedListener;

public class WeatherProvider extends AppWidgetProvider
```

```
{
   // размер выборки для изображения прогноза Bitmap
   private static final int BITMAP_SAMPLE_SIZE = 4;

```

Методы onUpdate, getZipcode и onReceive класса WeatherProvider

Метод onUpdate (листинг 14.60, строки 27–32) в ответ на широковещательные сообщения выполняет сравнение с константой ACTION_APPWIDGET_UPDATE класса AppWidgetManager. В подобном случае вызывается метод startUpdateService (см. листинг 14.60) для обновления прогноза погоды.

Метод getZipcode (строки 35–48) возвращает почтовый индекс предпочтительного города из объекта SharedPreferences приложения.

Метод onReceive (строки 51–61) вызывается в случае, если WeatherProvider принимает широковещательный Intent. Выполняется проверка на предмет соответствия между действием данного объекта Intent и WeatherViewerActivity.WIDGET_UPDATE_BROADCAST. Широковещательная рассылка объекта Intent класса WeatherViewerActivity выполняется в случае изменения предпочтительного города. В результате виджет приложения может обновлять соответствующим образом прогноз погоды. Для отображения прогноза погоды для нового города запускается служба startUpdateService.

Листинг 14.60. Методы onUpdate, getZipcode и onReceive класса WeatherProvider

```
26   // обновляет все установленные виджеты Weather App
27   @Override
28   public void onUpdate(Context context,
29      AppWidgetManager appWidgetManager, int[] appWidgetIds)
30   {
31      startUpdateService(context); // запуск новой службы WeatherService
32   }  // конец определения метода onUpdate
33
34   // получение сохраненного почтового индекса для этого виджета приложения
35   private String getZipcode(Context context)
36   {
37      // получение SharedPreferences для приложения
38      SharedPreferences preferredCitySharedPreferences =
39         context.getSharedPreferences(
40         WeatherViewerActivity.SHARED_PREFERENCES_NAME,
41         Context.MODE_PRIVATE);
42
43      // получение почтового индекса для предпочтительного города
        // из SharedPreferences
44      String zipcodeString = preferredCitySharedPreferences.getString(
45         WeatherViewerActivity.PREFERRED_CITY_ZIPCODE_KEY,
46         context.getResources().getString(R.string.default_zipcode));
47         return zipcodeString; // return the ZIP code string
48   }  // конец определения метода getZipcode
49
50   // вызывается, если данный AppWidgetProvider получает
     // широковещательный Intent
```

продолжение ⇨

Листинг 14.60 *(продолжение)*

```
@Override
public void onReceive(Context context, Intent intent)
{
   // если предпочтительный город был изменен в приложении
   if (intent.getAction().equals(
      WeatherViewerActivity.WIDGET_UPDATE_BROADCAST_ACTION))
      {
         startUpdateService(context); // прогноз для нового города
   }  // конец блока if
   super.onReceive(context, intent);
}  // конец описания метода onReceive

```

Метод startUpdateService класса WeatherProvider

Метод startUpdateService (листинг 14.61) запускает новую службу IntentService, имеющую тип WeatherService (листинг 14.62), которая обновляет прогноз для виджета приложения в фоновом потоке.

Листинг 14.61. Метод startUpdateService класса WeatherProvider

```
// запуск новой службы WeatherService, обновляющей
// прогноз, отображаемый виджетом приложения
private void startUpdateService(Context context)
{
   // создание нового объекта Intent, запускающего
   // службу WeatherService
   Intent startServiceIntent;
   startServiceIntent = new Intent(context, WeatherService.class);

   // включение почтового индекса в качестве дополнения Intent
   startServiceIntent.putExtra(context.getResources().getString(
      R.string.zipcode_extra), getZipcode(context));
      context.startService(startServiceIntent);
}  // конец определения метода startUpdateService

```

Вложенный класс WeatherService класса WeatherProvider

Служба IntentService класса WeatherService (листинг 14.62) выбирает информацию из веб-службы WeatherBug и обновляет компоненты View виджета приложения. Конструктор службы IntentService (строки 80–83) принимает строку String, используемую для именования рабочего потока (Thread) службы. Эта строка используется для выполнения отладки. Метод onHandleIntent (строки 89–101) вызывается после запуска службы WeatherService. Мы получаем объект Resources из контекста приложения, а также получаем почтовый индекс от объекта Intent, который запустил службу. Затем запускается метод ReadLocationTask, выполняющий считывание информации о местоположении для данного почтового индекса.

Листинг 14.62. Вложенный класс WeatherService класса WeatherProvider

```
// обновляет виджет приложения Weather Viewer
public static class WeatherService extends IntentService
implements ForecastListener
{
   public WeatherService()
   {
      super(WeatherService.class.toString());
   }  // конец определения конструктора WeatherService

   private Resources resources;   // ресурсы приложения
   private String zipcodeString;  // почтовый индекс
                                  // предпочтительного города
   private String locationString; // местоположение предпочтительного
                                  // города

   @Override
   protected void onHandleIntent(Intent intent)
   {
      resources = getApplicationContext().getResources();

      zipcodeString = intent.getStringExtra(resources.getString(
         R.string.zipcode_extra));

      // загрузка в фоновый поток информации о местоположении
      new ReadLocationTask(zipcodeString, this,
         new WeatherServiceLocationLoadedListener(
         zipcodeString)).execute();
   }   // конец определения метода onHandleIntent

```

Метод onForecastLoaded вложенного класса WeatherService

Метод onForecastLoaded (листинг 14.63) вызывается после завершения чтения информации о погоде из веб-службы WeatherBug, которое выполняет AsyncTask. Сначала проверяется, не имеет ли возвращаемый Bitmap значение null. Положительный результат проверки означает, что ReadForecastTask не может вернуть корректные данные, поэтому просто отображается сообщение Toast. Если же результат проверки отрицателен, создается новый объект PendingIntent (строки 118–120), который будет использован для запуска WeatherViewerActivity после загрузки пользователем виджета приложения. Объект PendingIntent представляет объект Intent и действие, выполняемое вместе с этим объектом Intent. Причина использования объекта PendingIntent в данном случае — его способность передаваться между процессами.

При обновлении виджета приложения из AppWidgetProvider не происходит непосредственного обновления компонентов View виджета приложения. Виджет приложения фактически обновляется в отдельном процессе от AppWidgetProvider. Передача данных между этими двумя объектами осуществляется с помощью объекта класса RemoteViews. Для разметки виджета приложения создается новый объект RemoteViews (строки 123–124). Затем объект PendingIntent передается методу setOnClickPendingIntent

класса remoteView (строки 127–128), который регистрирует объект PendingIntent виджета приложения. Это происходит после касания значка виджета приложения с целью запуска приложения Weather Viewer. Указывается ID разметки для корневого компонента View в иерархии объектов View виджета приложения. Компоненты TextView виджета приложения обновляются путем передачи ID ресурса каждого TextView и желаемого текста методу setTextViewText класса RemoteView. Компонент ImageView отображает изображения с помощью метода setImageViewBitmap класса RemoteView. Создается новый объект ComponentName (строки 154–155), представляющий компонент приложения WeatherProvider. Формируется ссылка на AppWidgetManager приложения с помощью статического метода getInstance (строка 158). Чтобы применить изменения, внесенные в компоненты RemoteViews, к компонентам View виджета приложения, объекты ComponentName и RemoteViews передаются методу updateAppWidget класса AppWidgetManager (строка 161).

Листинг 14.63. Метод onForecastLoaded вложенного класса WeatherService

```
103  // получает информацию о погоде из ReadForecastTask
104  @Override
105  public void onForecastLoaded(Bitmap image, String temperature,
106     String feelsLike, String humidity, String precipitation)
107  {
108     Context context = getApplicationContext();
109
110     if (image == null) // отсутствуют возвращаемые данные
111     {
112        Toast.makeText(context, context.getResources().getString(
113           R.string.null_data_toast), Toast.LENGTH_LONG);
114        return; // выход перед обновлением прогноза
115     }  // конец блока if
116
117     // создание объекта PendingIntent, используемого
        // для запуска WeatherViewerActivity
118     Intent intent = new Intent(context, WeatherViewerActivity.class);
119     PendingIntent pendingIntent = PendingIntent.getActivity(
120        getBaseContext(), 0, intent, 0);
121
122     // получение компонентов RemoteViews для виджета приложения
123     RemoteViews remoteView = new RemoteViews(getPackageName(),
124        R.layout.weather_app_widget_layout);
125
126     // настройка PendingIntent для запуска виджета приложения
        // после щелчка пользователя
127     remoteView.setOnClickPendingIntent(R.id.containerLinearLayout,
128        pendingIntent);
129
130     // отображение информации о местоположении
131     remoteView.setTextViewText(R.id.location, locationString);
132
133     // отображение температуры
134     remoteView.setTextViewText(R.id.temperatureTextView,
```

```
         temperature + (char)0x00B0 + resources.getString(
         R.string.temperature_unit));

      // отображение температуры "по ощущениям"
      remoteView.setTextViewText(R.id.feels_likeTextView, feelsLike +
         (char)0x00B0 + resources.getString(R.string.temperature_unit));

      // отображение влажности воздуха
         remoteView.setTextViewText(R.id.humidityTextView, humidity +
         (char)0x0025);

      // отображение вероятности осадков
      remoteView.setTextViewText(R.id.precipitationTextView,
         precipitation + (char)0x0025);

      // отображение картинки прогноза
      remoteView.setImageViewBitmap(R.id.weatherImageView, image);

      // получение Component Name для идентификации обновляемого виджета
      ComponentName widgetComponentName = new ComponentName(this,
         WeatherProvider.class);

      // получение глобального AppWidgetManager
      AppWidgetManager manager = AppWidgetManager.getInstance(this);

      // обновление AppWdiget Weather
      manager.updateAppWidget(widgetComponentName, remoteView);
   } // конец определения метода onForecastLoaded

```

Класс WeatherServiceLocationLoadedListener класса WeatherService

Класс WeatherServiceLocationLoadedListener (листинг 14.64) получает информацию о местоположении из веб-службы WeatherBug с помощью AsyncTask. С помощью конструктора onLocationLoaded (строки 177–202) создается строка String, использующая возвращенные данные, затем вызывается новый метод ReadForecastTask, начинающий считывать сведения о текущей погоде для предпочтительного города. С помощью метода setSampleSize класса ReadForecastTask устанавливается размер выборки данных, используемых для отображения иллюстрации с помощью объекта Bitmap. Обратите внимание на ограничения размеров объектов Bitmap, которые могут отображаться с помощью компонентов RemoteView.

Листинг 14.64. Класс WeatherServiceLocationLoadedListener класса WeatherService

```
164   // получает сведения о местоположении от фоновой задачи
165   private class WeatherServiceLocationLoadedListener
166   implements LocationLoadedListener
167   {
168      private String zipcodeString; // почтовый индекс для просмотра
169
```

продолжение ⇨

Листинг 14.64 *(продолжение)*

```
      // создание нового класса WeatherLocationLoadedListener
      public WeatherServiceLocationLoadedListener(String zipcodeString)
      {
         this.zipcodeString = zipcodeString;
      }  // конец определения класса WeatherLocationLoadedListener

      // вызывается после загрузки информации о местоположении
      @Override
      public void onLocationLoaded(String cityString,
         String stateString, String countryString)
      {
         Context context = getApplicationContext();

         if (cityString == null) // если нет возвращаемых данных
         {
            Toast.makeText(context, context.getResources().getString(
               R.string.null_data_toast), Toast.LENGTH_LONG);
            return; // выход перед обновлением прогноза
         }  // конец блока if

         // отображение возвращаемой информации в TextView
         locationString = cityString + " " + stateString + ", " +
            zipcodeString + " " + countryString;

         // запуск новой ReadForecastTask
         ReadForecastTask readForecastTask = new ReadForecastTask(
            zipcodeString, (ForecastListener) WeatherService.this,
            WeatherService.this);

         // ограничение размера Bitmap
         readForecastTask.setSampleSize(BITMAP_SAMPLE_SIZE);
         readForecastTask.execute();
         } // конец определения метода onLocationLoaded
       }// конец определения класса  WeatherServiceLocationLoadedListener
     } // конец определения класса WeatherService
}  // конец определения WeatherProvider
```

14.6. Резюме

В этой главе вашему вниманию были представлены приложение Weather Viewer и вспомогательный виджет приложения. Это приложение использует ряд новых функций, появившихся в Android 3.x.

Вы освоили применение фрагментов для создания и управления отдельными «порциями» графического интерфейса приложения. Были использованы подклассы DialogFragment и ListFragment класса Fragment для создания устойчивого интерфейса пользователя и обеспечения использования преимуществ экрана планшета. Вы узнали о том, что каждый объект Fragment имеет жизненный цикл и должен поддерживаться родительским объектом Activity. С помощью FragmentManager выполнялось управление

объектами Fragments и FragmentTransaction, обеспечивая добавление, удаление и выполнение переходов между фрагментами.

Панель действий Android 3.x, отображаемая в верхней части экрана, использовалась для отображения меню параметров приложения и элементов навигационной панели с вкладками. Использовалась обработка событий длительных нажатий, в результате чего пользователю предоставлялась возможность выбирать город в качестве предпочтительного или удалять город из списка. Приложение также использовало JsonReader для чтения объектов JSON, содержащих данные о погоде, из веб-служб WeatherBug.

Был создан вспомогательный виджет приложения (путем расширения класса AppWidgetProvider), предназначенный для отображения текущей погоды для предпочтительного города, выбранного пользователем. Для запуска приложения после касания пользователем значка виджета использовался объект PendingIntent. Если пользователь изменял предпочтительный город, приложение с помощью объекта Intent транслировало изменения виджету приложения.

Оставайтесь на связи с Deitel & Associates, Inc

Мы надеемся, что вы получили не меньшее удовольствие от чтения книги, чем мы от работы над ней. Мы ждем ваши отзывы. Присылайте ваши вопросы, комментарии, предложения и замечания по адресу deitel@deitel.com. Проверяйте наш постоянно растущий список Resource Centers, связанных с Android, на сайте www.deitel.com/ResourceCenters.html. Найдите нас в Фейсбуке (www.deitel.com/deitelfan) и в Твиттере (@deitel).

П. Дейтел, Х. Дейтел, Э. Дейтел, М. Моргано

Android для программистов: создаём приложения

Перевел с английского А. Сергеев

Заведующий редакцией	*А. Кривцов*
Ведущий редактор	*Ю. Сергиенко*
Научный редактор	*А. Пасечник*
Художественный редактор	*Л. Адуевская*
Корректор	*В. Листова*
Верстка	*Л. Харитонов*

ООО «Мир книг», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Подписано в печать 08.08.12. Формат 70x100/16. Усл. п. л. 45,150. Тираж 2000. Заказ
Отпечатано с готовых диапозитивов в ГППО «Псковская областная типография».
180004, Псков, ул. Ротная, 34.